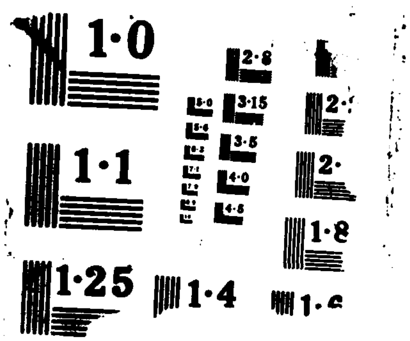
AD-A193 309 E&V (EVALUATION AND VALIDATION) MANUAL VERSION 1.0(U) 1/4
ADA JOINT PROGRAM OFFICE ARLINGTON VA 29 DEC 87
TASC-TR-5234-3

UNCLASSIFIED F/G 12/5 NL


1·0
2·8
3·15
3·5
4·0
1·1
1·8
1·25
1·4

AD-A193 309

DTIC FILE COPY

DoD Symbol

E&V Symbol

E&V REFERENCE MANUAL

VERSION 1.0
29 December 1987

Approved for public release; distribution is unlimited

The Task for the Evaluation and Validation (E&V) of Ada Programming Support Environments (APSEs) is sponsored by the Ada Joint Program Office.

DTIC
ELECTE
APR 04 1988
S D
E

TASC No. TR-5234-3

88 4 4 055

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETEING FORM

1. REPORT NUMBER TASC No. TR-5234-3 — 2. GOVT ACCESSION NO. ADA193309	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* E&V Reference Manual, Version 1.0	5. TYPE OF REPORT & PERIOD COVERED Reference Manual
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Ada Joint Program Office	8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION AND ADDRESS Ada Joint Program Office, 3E114, The Pentagon, Washington, D.C.20301-3081	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office 3E 114, The Pentagon Washington, DC 20301-3081	12. REPORT DATE 29 December 1987
	13. NUMBER OF PAGES 341
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* Ada Joint Program Office	15. SECURITY CLASS *(of this report)* UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20. If different from Report)*

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS *(Continue on reverse side if necessary and identify by block number)*

Ada Programming language, Ada Joint Program Office, AJPO, Ada Programming Support Environments, APSE, MIL-STD-1815A

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See Attached

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

EXECUTIVE SUMMARY

The Ada community, including government, industry, and academic personnel, needs the capability to assess APSEs (Ada Programming Support Environments) and their components and to determine their conformance to applicable standards (e.g., DoD-STD-1838, the CAIS standard). The technology required to fully satisfy this need is extensive and largely unavailable; it ~~cannot be~~ acquired by a single government-sponsored, professional society-sponsored, or private effort. The purpose of the APSE Evaluation and Validation (E&V) task is to provide a focal point for addressing the need by:

(1) Identifying and defining specific technology requirements;

(2) Developing selected elements of the required technology;

(3) Encouraging others to develop some elements; and

(4) Collecting information describing existing elements.

This information will be made available to DoD components, other government agencies, industry and ~~academia~~.

The purpose of ~~the~~ E&V Reference Manual (this document) is to provide information that will help users to:

(1) Gain an overall understanding of APSEs and approaches to their assessment;

(2) Find useful reference information (e.g., definitions) about specific elements and relationships between elements; and

(3) Find criteria and metrics for assessing tools and APSEs, and techniques for performing such assessment.

The latter are to be found (or referenced) in a companion document called the E&V Guidebook.

Chapter 4 and later chapters are "formal chapters" built around a standard format and formal grammar. Each of the formal chapters corresponds to one index of an overall E&V Classification Schema. The schema adopts a relational model of the subject and process of E&V. This model will allow the user to arrive at E&V techniques through many different paths, and provides a means to extract useful information along the way.

Yearly updates and extensions to this manual are planned. Therefore, comments and suggestions are welcome. Please send comments electronically (preferred) to szymansk@ajpo.sei.cmu.edu or by regular mail to Mr. Raymond Szymanski, AFWAL/AAAF, Wright Patterson AFB, OH 45433-6543.

Accession For
NTIS GRA&I ☒
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes

Dist	Avail and/or Special
A-1	

INSPECTED 4

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

1. INTRODUCTION

1.1 PURPOSE OF THE MANUAL

This document is a product of the Ada Programming Support Environment (APSE) Evaluation and Validation (E&V) Task sponsored by the Ada Joint Program Office. It is one of a pair of companion documents known as the E&V Reference System, consisting of:

- E&V Reference Manual
- E&V Guidebook.

The purpose of the Reference Manual is to provide a collection of information to support a variety of users. The collection is organized in accordance with a Classification Schema described in Chapter 2. It should help users to:

- Gain an overall understanding of APSEs and approaches to the assessment of APSE performance, quality and conformance to applicable standards.
- Find useful reference information, such as definitions of specific elements of the Classification Schema, and relationships between elements.
- Find criteria and metrics for assessing specific components, combinations of components and "whole APSEs," and locate relevant E&V techniques.

The Reference Manual includes many "pointers" to sections in the Guidebook and other documents which describe E&V techniques. Figure 1.1-1 illustrates the relationship between the Reference Manual and the Guidebook. Figures 1.1-2 and 1.1-3 illustrate the types of information to be extracted from each document. Chapter 2 provides a more detailed description of the structure and uses of the Reference System.

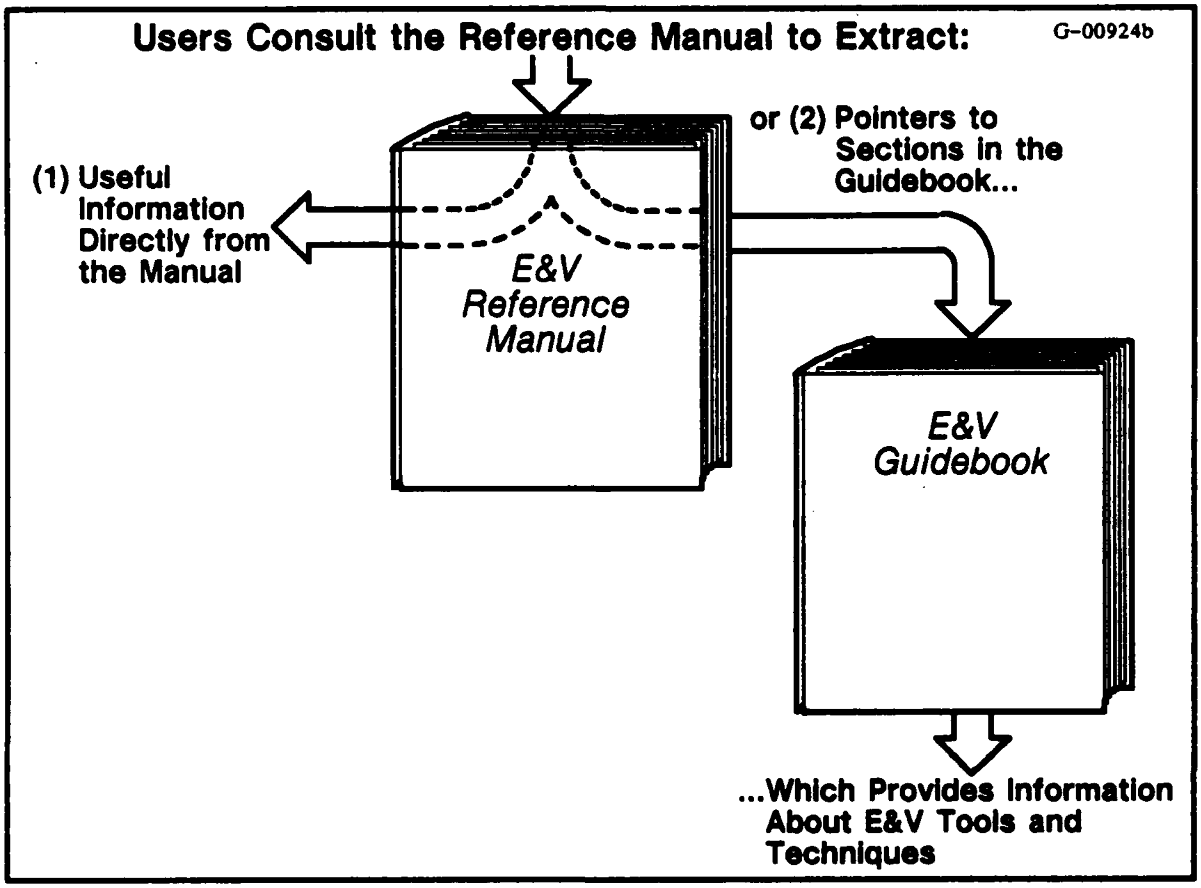


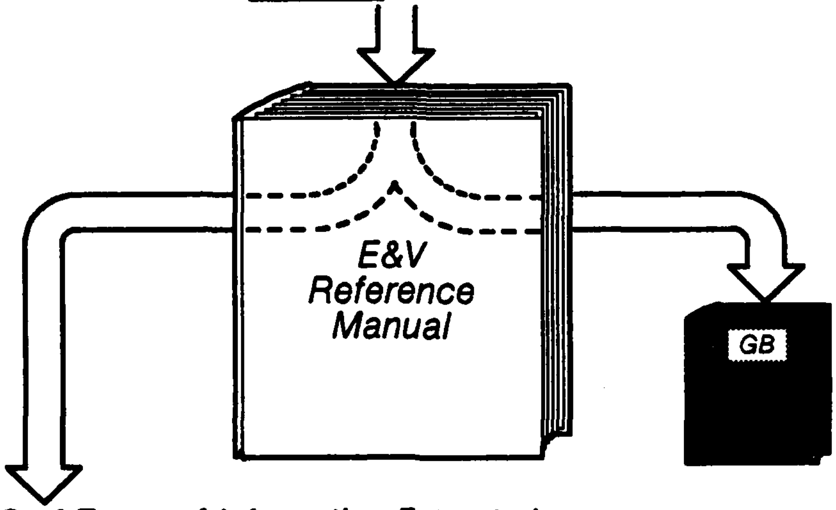
Figure 1.1-1 Uses of the Reference System

1.2 USERS OF THE MANUAL

Classes of people who are expected to be users of this manual are listed in Table 1.2-1. They are described in terms of their relationships to deliverable software, tools, APSEs, and APSE E&V technology. They may be associated with Government, industry, or academia. The table was adapted from material in the report of the E&V Workshop [@ E&V Workshop 1984].*

*The format used for references is associated with the "formal grammar" used beginning with Chapter 4. See further explanations in Chapter 2 and Appendix C.

Users consult <u>Indexes</u> in the Reference Manual G-00925b

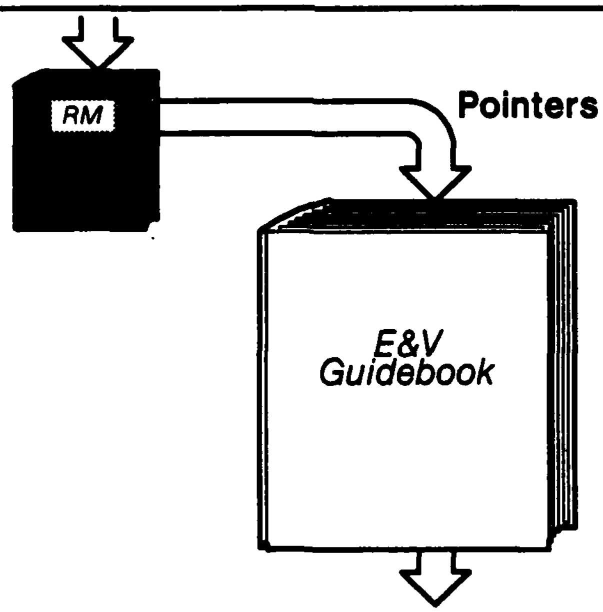


<u>EXAMPLES</u> of Types of Information Extracted:

- **Organization of The Indexes**
  <u>Hierarchical Structure</u> and a Numbering System
  (e.g., The Function Index . . . see Fig. 2.1-2)
- **Names and Descriptions of Specific Elements within the Indexes**
  (e.g., The Compilation Function . . . see Figs. 2.1-2 and 2.1-3)
- **Cross References Between Elements of the Indexes**
  (e.g., The Function "Compilation" is Cross Referenced to Phases and Tools . . . see Fig. 2.4-1)

Figure 1.1-2 Examples of Reference Manual Information

G-00926a



<u>EXAMPLES</u> of Types of Information Extracted:

- **Description of Evaluation (E) Tools and Their Uses** (e.g., see Section 2.4)
- **Description of Validation (V) Suites**
- **References to Other Documents Describing E or V Techniques**
- **Synopses of Other Documents That Cover Multiple Techniques**

Figure 1.1-3 Examples of Guidebook Information

While the manual is designed to be of use to people of all the classes listed, the primary users are expected to be the APSE/tool users and the E&V technology users. These are people with highly technical backgrounds and technical/managerial interests in evaluating and selecting tools and APSEs. This expectation has strongly influenced the structure and style of the manual. The primary users are likely to consult both the Reference Manual and the Guidebook. Many of the other classes of users listed in Table 1.2-1 are likely to consult the Reference Manual only.

TABLE 1.2-1
REFERENCE MANUAL USERS

CLASS	DESCRIPTION
Software Acquisition Personnel	Government program officers and commercial program managers who let contracts for software development
APSE/Tool Users	Project managers, librarians, system engineers, software engineers
APSE/Tool Builders	Environment/tool designers, implementers, managers, marketing personnel
E&V Technology Users	Government, commercial and university personnel applying E&V technology
E&V Technology Builders	Anyone developing E&V assessment techniques (E&V Task contractors, etc)
Investors	Anyone funding development or use of E&V technology (Congress, AJPO, corporations, etc.)

1.3 BACKGROUND

In June 1983 the Ada Joint Program Office (AJPO) proposed the formation of the E&V Task and a tri-service E&V Team, with the Air Force designated as lead

service. In October 1983 the Air Force officially accepted responsibility as lead service and designated the Air Force Wright Aeronautical Laboratories (AFWAL) at Wright Patterson Air Force Base as lead organization. In April 1984 an E&V Workshop was held at Airlie, Virginia. The purpose of the workshop was to solicit participation of industry representatives in the E&V Task. Many of the participants in the workshop have chosen to remain involved as Distinguished Reviewers, and additional industry participants have subsequently become involved in E&V Team activities.

The E&V Team publishes an annual public report. The following paragraph is quoted from the 1987 version [@E&V Report 1987] of the report:

> "The Ada community, including government, industry, and academic personnel, needs the capability to assess APSEs (Ada Programming Support Environments) and components and to determine their conformance to applicable standards (e.g., DoD-STD-1838, the CAIS standard). The technology required to fully satisfy this need is extensive and largely unavailable; it cannot be acquired by a single government-sponsored, professional society-sponsored, or private effort. The purpose of the APSE Evaluation and Validation (E&V) task is to provide a focal point for addressing the need by (1) identifying and defining specific technology requirements, (2) developing selected elements of the required technology, (3) encouraging others to develop some elements, and (4) collecting information describing existing elements. This information will be made available to DoD components, other government agencies, industry, and academia."

The team public reports contain much additional information for the interested reader. Three competitive contracts have been awarded under the E&V task. These are:

- Technical Support contract - awarded June 1985
- Ada Compiler Evaluation Capability (ACEC) contract - awarded February 1987
- CAIS Implementation Validation Capability (CIVC) contract - awarded May 1987.

The major purpose of the first of these contracts is to create and update elements of the E&V Reference System, including this manual. The purpose of the

second and third contracts is to create two specific elements (ACEC and CIVC) of the needed E&V technology.

1.4 ORGANIZATION OF THE MANUAL

Chapter 2 discusses the structure of the E&V Reference System (Reference Manual plus Guidebook) and the Classification Schema upon which that structure is based. Specific directions as to how to use the manual are also included.

Chapter 3 provides a general discussion of "whole APSE" issues, in which an APSE is viewed as more than the sum of its parts. Key whole-APSE attributes are discussed along with general approaches to whole-APSE assessment.

Chapter 4 and subsequent chapters are "formal chapters" built around a standard format and formal grammar. Each of the formal chapters corresponds to one index of the Classification Schema. The structure and use of these chapters are the focus of the material found in Chapter 2.

The appendices include a description of the formal grammar, a glossary of acronyms and abbreviations, a document citation list, and a composite index.

2. USE OF THE REFERENCE SYSTEM

This chapter provides a step-by-step explanation of how to use the E&V Reference System and the Classification Schema upon which the system is based. Section 2.1 describes the organization of the material. Sections 2.2, 2.3, and 2.4 contain illustrations of uses of the system, presented in increasing levels of sophistication. Section 2.5 provides a global, conceptual view of the system framework. User A (Section 2.2) consults an index of the Reference Manual to find the description of a term that is an element of that index. User B (Section 2.3) consults an index to find an element and several cross references to related elements in another index. User C (Section 2.4) consults a combination of indexes to find references to sections in the Guidebook, which contain explanations of relevant Evaluation or Validation techniques. Brief definitions of several key words and expressions follow:

E&V	- Evaluation and Validation
Evaluation	- Assessing performance and quality
Validation	- Assessing conformance to a standard
E&V Reference System	- Two documents: the E&V Reference Manual and the E&V Guidebook
E&V Classification Schema	- A set of indexes that provide a framework for the E&V Reference Manual.

The schema was initially described in an earlier "E&V Classification Schema Report" [@E&V Classification], which was used as the starting point for the schema defined in this manual. Subsequent changes in the schema will be updated in future versions of this manual; the schema report will not be updated.

2.1 SYSTEM ORGANIZATION

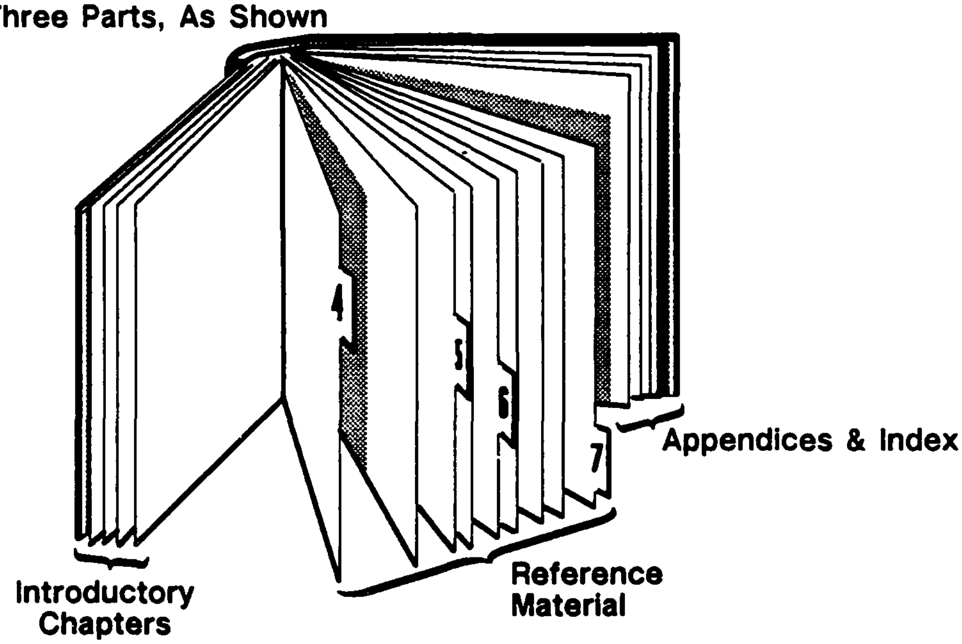
The entire reference system can be viewed as a structure of multiple indexes. For example, there is a function index and a life cycle phase index, among others. The structure is analogous to the card catalog system in a public library, with its author index, title index, and subject index. To use the card catalog in the library, you must first locate the card that corresponds to the author, title, or subject in which you are interested. Similarly, to use the Reference Manual, you first find the element(s) in which you are interested. The way to do this is to begin by looking at the Table of Contents or the Index at the back of the manual to locate the element(s) in the "formal chapters." The names of the indexes that are formal chapters within the Reference Manual are:

- Life Cycle Phases (Chapter 4)
- APSE Tool Categories (Chapter 5)
- Attributes (Chapter 6)
- Functions (Chapter 7).

Figures 2.1-1, 2.1-2, and 2.1-3 provide a pictorial view of the organization of the Reference Manual, particularly the structure of the reference material contained in the "middle section." The chapters of this middle section are organized in a consistent, formal manner, using a formal grammar (described in Appendix C). A typical chapter corresponds to one index of the schema. A typical index is organized as a hierarchical structure of elements. For every element there is a "text frame" that has (in general) three parts as shown in Fig. 2.1-3. The text frames are built using the formal grammar. (Details of the formal grammar need not concern the user. It was employed because of the possibility of a future on-line, electronic version of the system, supported by advanced updating and information retrieval techniques.)

G-00927b

The E&V Reference Manual Has Three Parts, As Shown

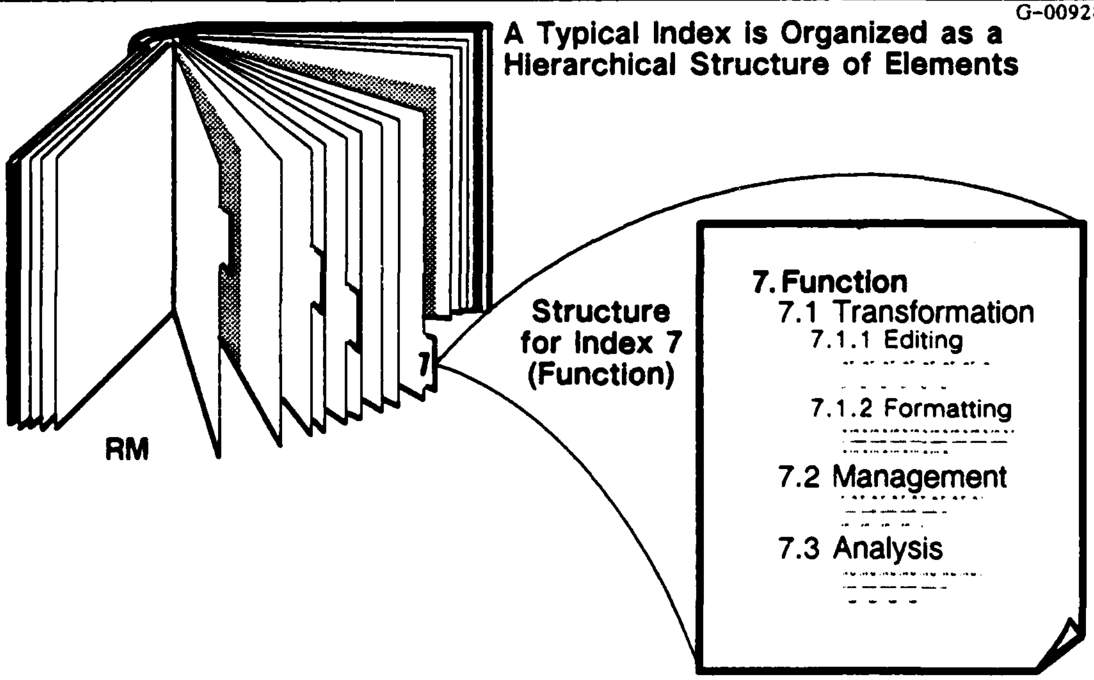


Typically a chapter in the "Reference Material" section corresponds, to one index of the Classification Schema. These chapters are organized in a consistent, formal manner.

Figure 2.1-1 Reference Manual Organization

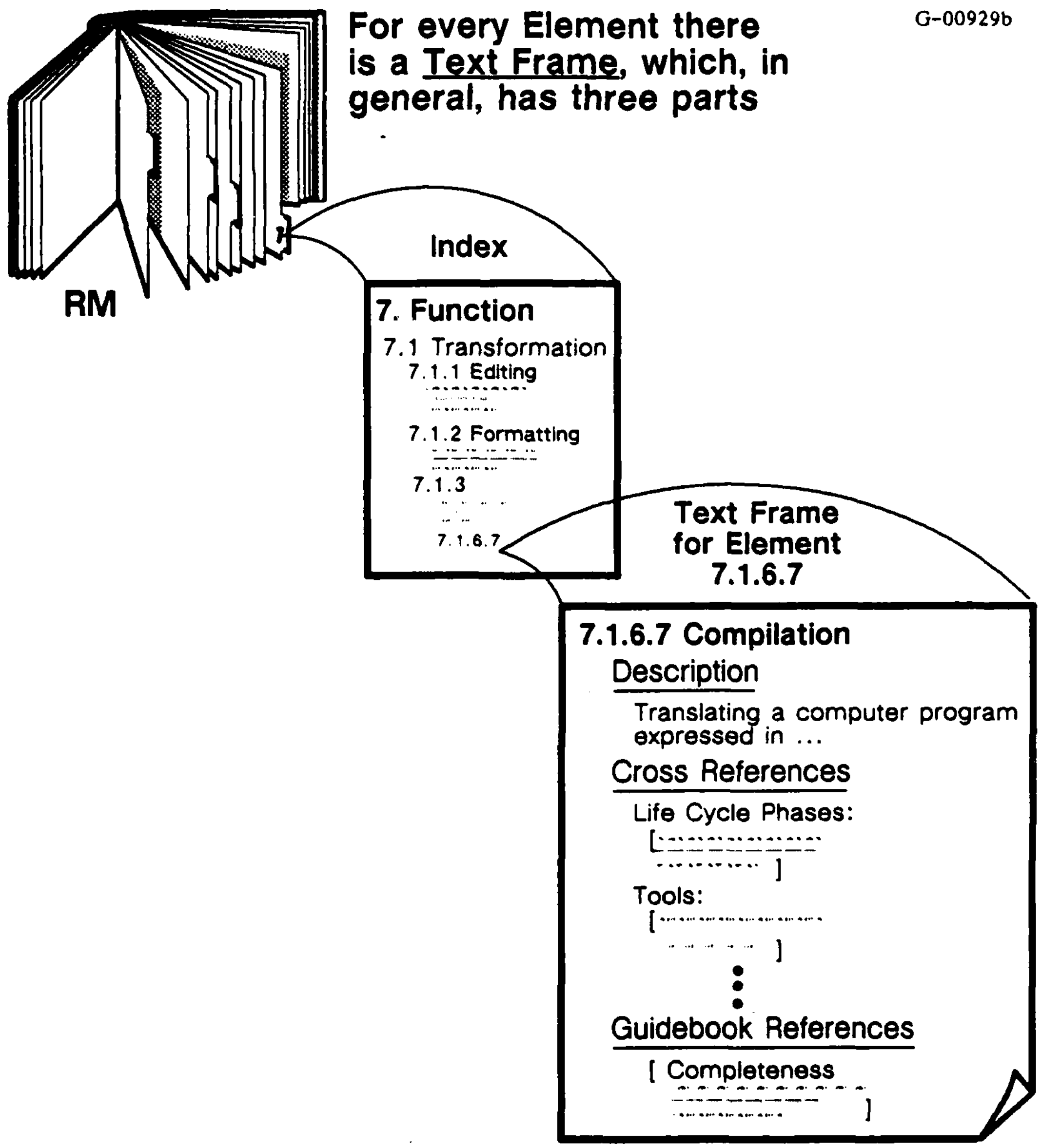
G-00928b

A Typical Index is Organized as a Hierarchical Structure of Elements



Each Element is Identified by a Number and a Name.

Figure 2.1-2 Reference Material Indexes



Description:	Is a brief definition of the element.
Cross References:	Identify significant relationships with other elements in the same index or other indexes of the Reference Manual.
Guidebook References:	Identify relationships involved in the statements of E&V objectives and point to sections in the Guidebook where information about how to satisfy objectives may be found.

Figure 2.1-3 Text Frames

2.2 DESCRIPTION: DIRECT REFERENCE

Our first example of the use of the Reference Manual is "User A" who consults the Attributes Index to find the description of the term "Storage Effectiveness." Figure 2.2-1 is a copy of Section (or Text Frame) 6.4.31. User A may find this frame, for example, by browsing through the Table of Contents or by looking up the term "Storage Effectiveness" in the main Index at the back of the manual. Note that the text frame contains Cross References and Guidebook References as well, but this need not concern User A, who simply seeks a description. The User A scenario is pictorially represented in Fig. 2.2-2, where the boxes are analogous to cabinets in a library card catalog system.

6.4.31 Storage Effectiveness

Description:

Those characteristics of the software which provide for minimum utilization of storage resources in performing functions. [@RADC 1985] The choice between alternative source code constructions based on those taking the minimum number of words of object code or in which the information-packing . . . is high. [@DACS 1979]

Cross References:

Software Quality Factors:
[Efficiency 6.1.1]

Beneficial Quality Factors:

Adverse Quality Factors:
[Maintainability 6.2.2,
Verifiability, Testability 6.2.3,
Transportability 6.3.4]

Guidebook References:

[Compilation	7.1.6.7, @GB: IDA Benchmarks	6.1;
Compilation	7.1.6.7, @GB: ACEC	6.2]

Figure 2.2-1 Sample Attributes Index Frame

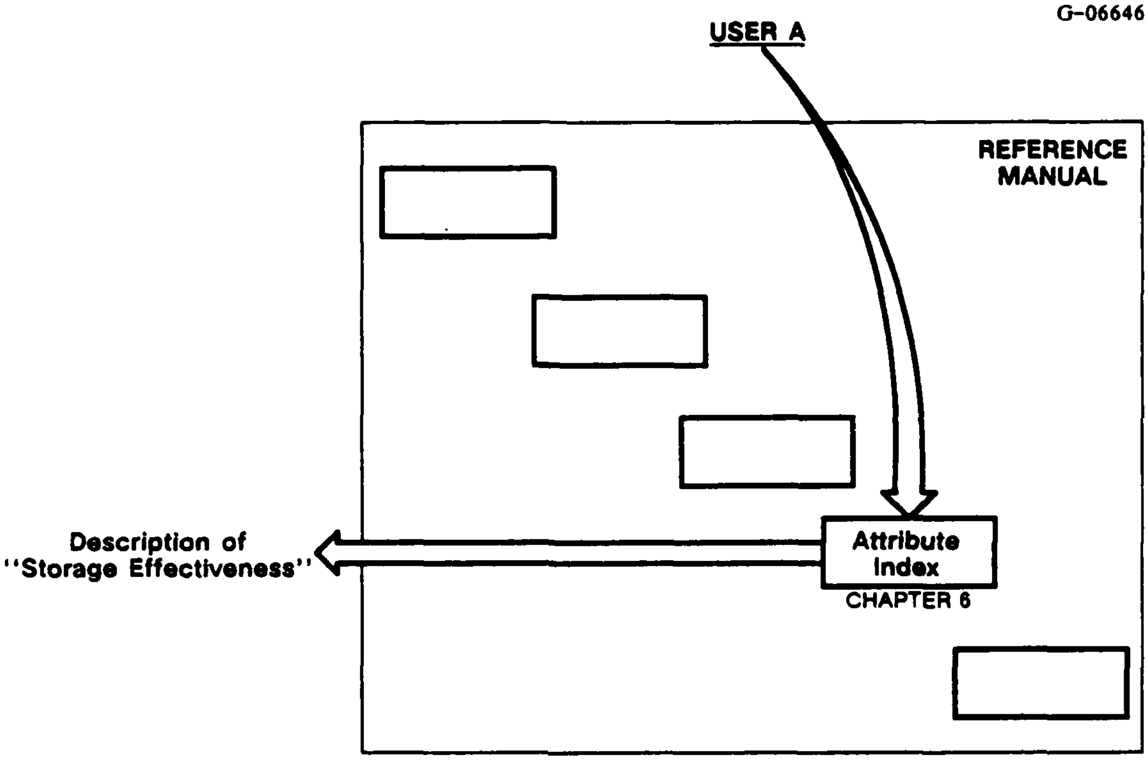


Figure 2.2-2 Example of *Direct Reference:* User A

2.3 CROSS REFERENCE

Our second example of use of the Reference Manual is "User B" who would like to know the names and descriptions of functions associated with a particular life cycle phase. User B first consults the Life Cycle Phases Index, Chapter 4, which outlines the various activities performed under the DoD-STD-2167 model [@DoD-STD-2167] for project development. Chapter 4 relates the life cycle phases to the functions typically found within an APSE and the deliverables that are produced in each of the phases. A sample Life Cycle Phases Index Frame is shown in Fig. 2.3-1. This frame represents the sixth phase [4.6 Coding and Unit Testing] and the second group of activities [4.6.2 Transformation] covered in each phase. The Software Development File is the deliverable for this group of activities. User B finds 16 functions that might be performed during this activity listed in the text frame. In each case another text frame is cross referenced in Chapter 7, the Functions Index, where more information about this function is found. The User B scenario is pictorially represented in Fig. 2.3-2.

4.6.2 Transformation	
Cross References:	
Deliverables:	
[Software Development File	(SDF)]
Functions:	
[Text Editing	7.1.1.1,
Predefined and User-Defined Forms	7.1.2.3,
Assembling	7.1.6.6,
Compilation	7.1.6.7,
Conversion	7.1.6.8,
Macro Expansion	7.1.9.9,
Structure Preprocessing	7.1.6.10,
Body Stub Generation	7.1.6.11,
Preamble Generation	7.1.6.12,
Linking/Loading	7.1.6.13,
Interpretation	7.1.6.14,
Requirements Reconstruction	7.1.7.2,
Program Generation	7.1.7.3,
Source Reconstruction	7.1.7.4,
Decompilation	7.1.7.5,
Disassembling	7.1.7.6]

Figure 2.3-1 Sample Life Cycle Phase Index Frame

2.4 GUIDEBOOK REFERENCE

Our third example of use of the Reference Manual is "User C" who would like to look up a function, learn what attributes are associated with it, and find evaluation techniques relevant to particular function-attribute pairs. User C first consults the Function Index, Chapter 7. A sample Function Index Frame [7.1.6.7 Compilation] is shown in Fig. 2.4-1. Functions are related to life cycle phases and tools. In the example, the function, Compilation, is related to three life cycle phases: Coding and Unit Testing, CSC Integration and Testing, and CSCI Testing. The user of the manual can refer to Chapter 4 to find more information about these phases. The Compiler is the tool that performs the function Compilation. Information about compilers can be found in Chapter 5. User C is interested in evaluation of the

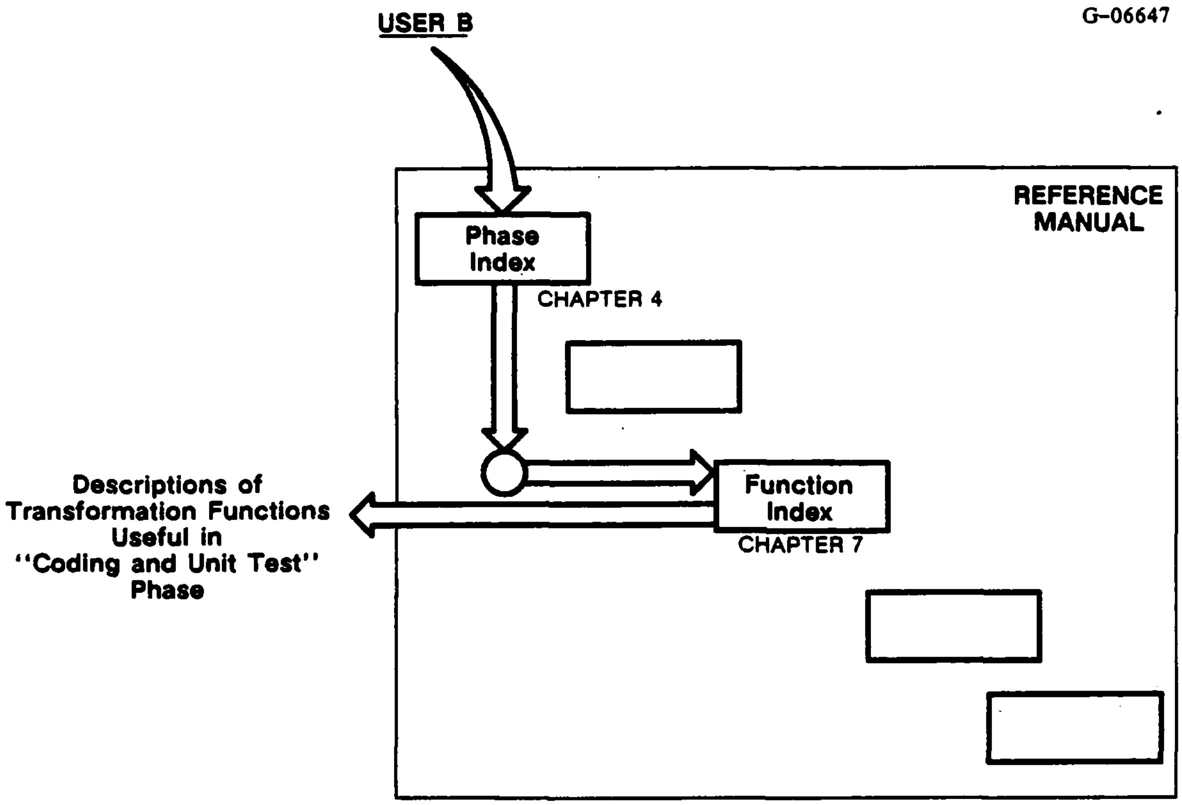


Figure 2.3-2 Example of Cross Reference: User B

compilation function with respect to various attributes. For example, the function-attribute pair Compilation-Processing Effectiveness is represented by the fourth and fifth items under Guidebook References in this text frame. This pair points to sections in the Guidebook called IDA Benchmarks and ACEC which provide additional information on these two E&V techniques. A user of the Reference Manual can find references to these two techniques in the Attribute Index or the Function Index. The User C scenario is pictorially represented in Fig. 2.4-2.

7.1.6.7 Compilation

Description:

Translating a computer program expressed in a procedural or problem-oriented language into object code. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

[Compiler	5.3.3]

Guidebook References:

[Capacity	6.4.6,	@GB: IDA Benchmarks	6.1;
Completeness	6.4.9,	@GB: ACVC	8.1;
Power	6.4.21,	@GB: Compilation Checklist	5.1.3;
Processing Effectiveness	6.4.22,	@GB: IDA Benchmarks	6.1;
Processing Effectiveness	6.4.22,	@GB: ACEC	6.2;
Storage Effectiveness	6.4.31,	@GB: IDA Benchmarks	6.1;
Storage Effectiveness	6.4.31,	@GB: ACEC	6.2]

Figure 2.4-1 Sample Functions Index Frame

G-06648

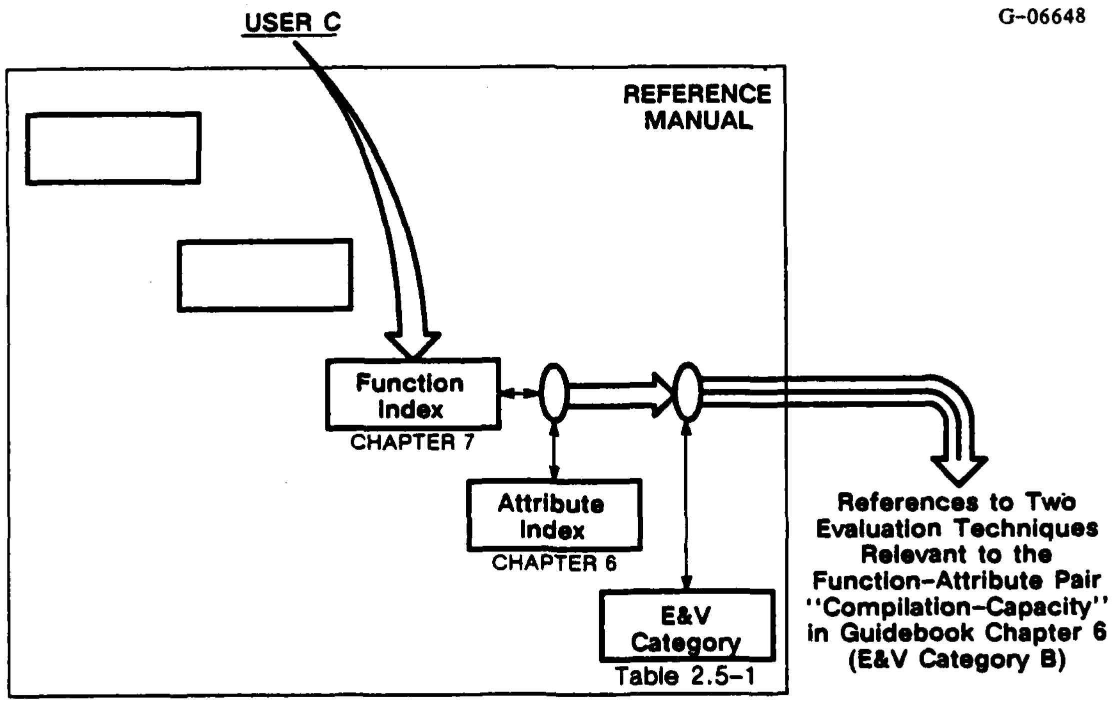


Figure 2.4-2 Example of Guidebook Reference: User C

2.5 REFERENCE FRAMEWORK

Figure 2.5-1 depicts the Classification Schema as an internal framework for the Reference Manual. The schema provides paths, within and between indexes, that users can follow to extract information directly or to find sections in the Guidebook that describe elements of E&V technology. The figure also indicates the direct relationships, that is, types of cross references and Guidebook references featured in the Reference Manual.

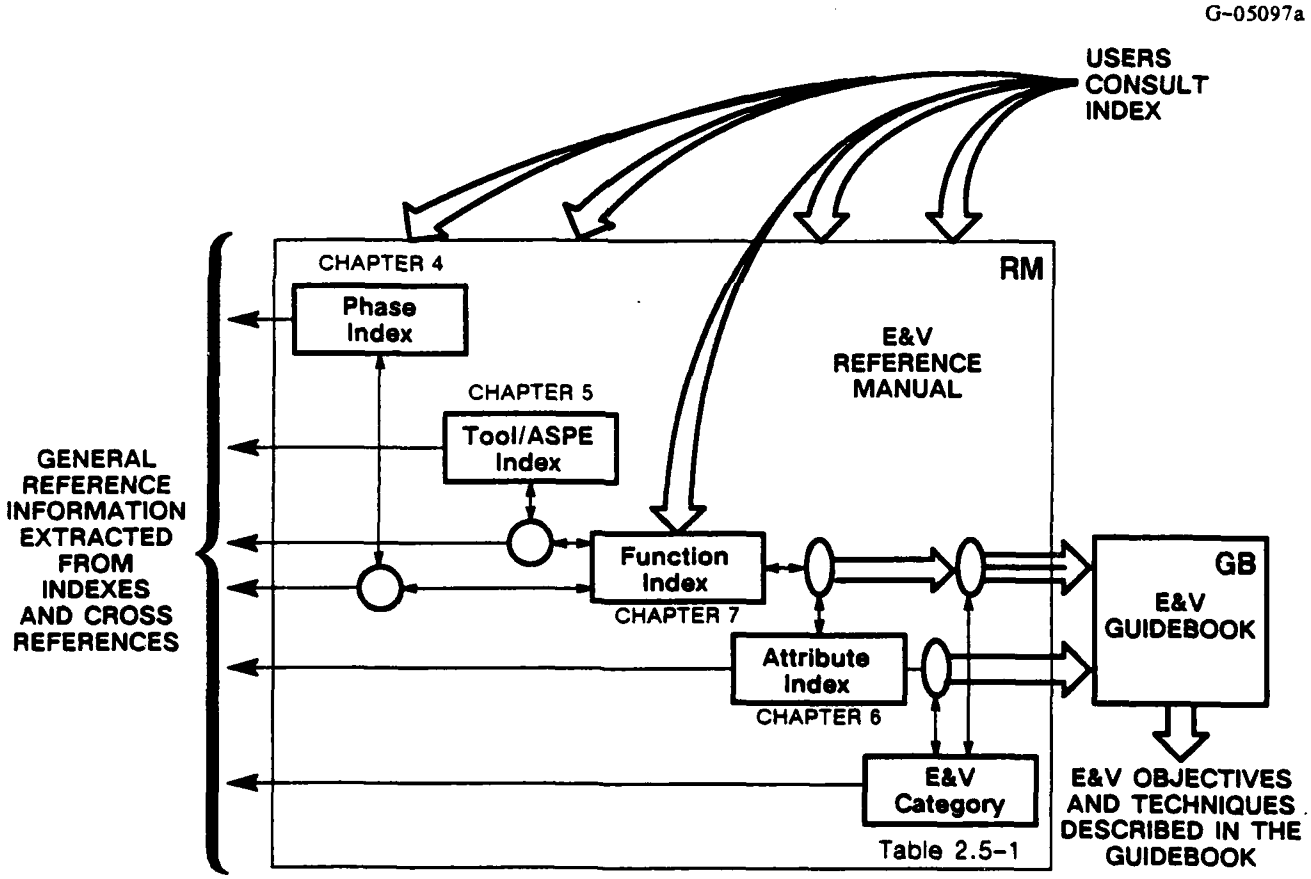


Figure 2.5-1 The Schema as Framework for the Reference Manual

The Function Index is seen to be directly related to all of the other indexes. Thus, it is drawn in a central position in the diagram, indicating a kind of central importance. Attributes, also, play a central role in the overall E&V process, in two ways. First, many assessment objectives are defined in terms of function-attribute pairs, such as compilation-efficiency and editing-power. Second, other attributes (those not "pairing-up" with functions) represent factors or criteria by which APSEs or APSE components are assessed, independent of the functions performed. Examples of the latter are: maintainability, interoperability, maturity, and cost.

Besides the direct relationships indicated in Fig. 2.5-1, indirect relationships can also be useful and are constructed by combining two or more direct relationships. For example, since phases and functions are related and functions and tools are related, it is possible to determine the relationships between phases and tools. It should be noted that not all such constructions will be useful. Such a useless relationship might be life cycle phases and attributes related via function.

The conceptual structure pictured in Fig. 2.5-1 has an open-ended quality in several ways. First, there is no fixed number of indexes; more can be added as desired. As each new index is added a new section of the Reference Manual can be added, providing appropriate definitions and cross-references to other indexes. The new index would be represented by an additional block along the "main diagonal." Second, each individual index may have elements added to it as new understanding of the various aspects of APSEs and the E&V process is gained. The process of modifying the structure of an individual index in this way would take place within the boxes. Finally, the off-diagonal space, above and below the main diagonal, represents the notion that any two-way relationship may be included in the system. Thus, any section may be referenced by other sections, and any section may reference other sections. Also, any two indexes may be involved in a relationship of the type that defines an E&V objective or points to an E&V technique in the Guidebook. Although many of the potential combinations are not expected to be useful or relevant to E&V purposes, the structure permits the consideration of all possibilities.

The procedures described in the Guidebook are organized into chapters by the E&V Category of the procedure. The E&V Categories are described in Table 2.5-1. The two factors which determine what category is appropriate are

TABLE 2.5-1
E&V CATEGORIES

CATEGORY/ GUIDEBOOK CHAPTER	BASIS FOR ASSESSMENT	MECHANISM	CHARACTERIZATION
A/Chapter 5	Qualitative Judgment	Informal (e.g., Questionnaire)	Subjective Evaluation
B/Chapter 6	Metric	Test Suite	Objective Evaluation
C/Chapter 7	Conformance to a Standard	Informal	Intermediate
D/Chapter 8	Conformance to a Standard	Test Suite	Validation
E/Chapter 9	Conformance to a Standard	Purely Formal Test Suite	Formal Validation

whether a standard exists with which to measure the attribute and what the mechanism is to measure the degree to which the APSE or APSE component possesses the attribute. The categories assigned to a function-attribute pair or functionally independent attribute may change as E&V standards are identified or new techniques are developed where none previously existed. For that reason, techniques for a given attribute or function-attribute pair may span several E&V Categories. Category A through E techniques are found in Chapters 5 through 9 of the Guidebook, respectively.

3. WHOLE APSE ISSUES

This chapter is intended as a starting place for those interested in selecting or assessing an APSE considered as a "whole entity" that is "more than the sum of its parts." The chapter, therefore, cannot be organized in the index/text frame style of other chapters in this manual, because that style fits the view of an APSE as a collection of components or functions that can be evaluated one at a time and "added up." Here we take the opposite view -- that the APSE is a total system which serves a project team across an entire software development life cycle -- and that it should be evaluated in terms of overall project goals and team productivity, rather than in terms of individual atomic functions.

Although the technology of "whole APSE assessment" is (at this writing) very immature, there are many helpful materials in the open literature. This chapter may be considered a guide to that literature. It is organized to address the following types of questions:

What is an APSE?
How can whole APSEs be viewed?
What are the key whole APSE attributes?
How can whole APSEs be assessed?
Where can relevant information be found?

The first four questions are addressed in Sections 3.1 through 3.4, respectively. Various references to other sections of the E&V Reference Manual (RM), the E&V Guidebook (GB), and open-literature sources are distributed throughout the chapter.

3.1 APSE DEFINITIONS AND ALTERNATIVE NAMES

The acronym APSE stands for "Ada Programming Support Environment." The term "Programming Support Environment" is defined in the "IEEE Standard Glossary of Software Engineering Terminology" [@IEEE 1983] as

> "An integrated collection of tools accessed via a single command language to provide programming support capabilities throughout the software life-cycle. The environment typically includes tools for design, editing, compiling, loading, testing, configuration management, and project management."

Thus, one useful definition of an APSE is the above quotation along with the stipulation that there be at least one Ada compiler among the tools provided. A similar definition for the term IPSE, which stands for "Integrated Project Support Environment," is given [@Lehman and Turski 1987] as

> "An embodiment of software technology in a collection of tools for capture, representation, control, refinement, transformation, and other manipulation of project related information."

The second name and definition are broader than the first because they refer to total "project" support and information, rather than "programming" support and programs. The distinction between the two is highlighted not only in the Lehman and Turski paper, but in many others, including a survey paper by Houghton and Wallace [@Houghton and Wallace 1987]. The latter uses the following terms to characterize most existing environments:

- *Framing Environments*
- Programming Environments
- General Environments.

Framing environments concentrate on the early stages of the life cycle and tend to be methodology-specific. Programming environments concentrate on the latter part of the life cycle and are oriented toward programming, debugging, and testing. General environments contain basic tools that support all phases of the life cycle and tend to be methodology-free.

Both of the above definitions and discussion are limited in that they are tied to one, traditional view of an APSE. In this view an APSE (or IPSE) is seen as a collection of tools. The following section presents additional views, any of which may become increasingly important to those who wish to select or evaluate APSEs of the future.

Additional acronyms found in the literature on this same general topic are listed below, among others mentioned previously.

APSE - Ada Programming Support Environment
IPSE - Integrated Project Support Environment
PSE - Programming Support Environment
SDE - Software Development Environment
SEE - Software Engineering Environment

The term SDE, for example, has been used in the title of several ACM-sponsored conferences, and is the subject of an IEEE tutorial [@IEEE 1981]. All of the above are possible key words, under which may be found useful material on whole APSEs or whole APSE E&V, the subject of this chapter.

A definition of an APSE, based on the concept of a collection of tools, is limited in the sense that it describes only a portion of the total environment for software development. As discussed in the paper "The Ecology of Software Environments" [@Wasserman 1981a], the total environment or "surroundings" includes such things as the computer itself, peripheral storage and display devices, project management practices, organizational characteristics, and external constraints such as governmental directives and physical workspace factors. All of the above will influence the development and application of whole APSE evaluation techniques.

3.2 VIEWS OF AN APSE

Various ways of viewing an APSE are summarized briefly below. The views differ in terms of both <u>how</u> an APSE is seen -- that is, what sort of conceptual model does the viewer hold -- and <u>by whom</u> is the APSE viewed -- for example, the APSE user would typically have a different view than the APSE builder. The views presented are not necessarily mutually exclusive; an individual's view of a specific APSE may combine aspects of several of the following notions.

3.2.1 <u>APSE Viewed as a Collection of Tools</u>

This is the traditional view of programming support environments, and is consistent with the standard definition quoted in the previous section. An advantage of

taking this view is that it permits the viewer to characterize an APSE in terms of elements of a functional taxonomy, such as that provided in Chapter 7 [Functions 7]. The characterization can be stated in terms of yes/no answers to a long list of clearly defined questions -- is function x.y.z provided or is it not? A disadvantage of this view is that it may cause the viewer to neglect the crucial whole-APSE issues that are the subject of this chapter, and which are impossible to express in terms of a composition of individual low-level functions.

One version of this view is given in the Stoneman Report [@DoD 1980], which pictures a multi-layered, extensible collection built around an inner kernel (KAPSE) and a surrounding minimal (MAPSE) layer of essential tools. This version may have more relevance for APSE builders than for APSE users, since users need not necessarily be aware of the layer in which a particular tool resides.

3.2.2 APSE Viewed as a Methodology-Support System

In this view an APSE is seen as a system that supports a particular development methodology or a particular model of the software development process. It might be strongly tied to a standard set of deliverable products and phases such as those required by the U.S. Department of Defense [@DoD-STD-2167] for mission critical software. It might be based upon a coherent software development and maintenance methodology, such as that described in "Life-Cycle Support in the Ada Environment" [@McDermid and Ripken 1984]. The latter expresses the view that, "The purpose of a tool is to support a method by automating some aspect of the use or application of the method."

3.2.3 APSE Viewed as an Information Management System

In this view primary emphasis is placed on how project information is stored, retrieved, manipulated, and maintained over the entire life cycle. A key issue is the set of interfaces between tools and the project data base [@Houghton and Wallace 1987]; the data base may be considered a tool that is used by other tools and therefore is the interface between them. Another key issue is the control of access to information, and how this is affected by data structure models -- such as hierarchical, relational, transactional, etc. McDermid [@McDermid 1985] envisions

three generations of environments as follows. The first generation would be a conventional collection of Ada tools tied to a Unix-like environment. The second generation would be based on a data base schema that provides appropriate life cycle support, but has a static, non-adaptive structure. The third generation would provide a dynamic, adaptive information management system that can encapsulate any underlying data base schema.

Another key set of issues to be addressed are those specific to the Ada compilation process. A unique feature of the Ada language is that inter-unit dependencies exist at compilation time. Source code units can be compiled separately, but not independently. The implications of this feature are discussed further in Section 3.3.1 under Integrity.

3.2.4 APSE Viewed as a User-Oriented, Interactive System

This view emphasizes user interfaces and human factors. According to the authors of a survey paper [@Houghton and Wallace 1987], "Many systems that measure poorly in terms of human factors, including most traditional operating systems, have withstood the test of time. . . . The real users are the systems programmers or gurus." (But, in the future) "Software engineering environments . . . should not require a system expert as an interface between the software engineer and the environment." A conference overview paper [@Henderson 1987] sees "a trend toward the use of graphics." The desire for graphical interactive workstations is echoed in a paper on personal development systems [@Gutz et al 1981] with the words, "time sharing is an idea whose time is gone." An important set of issues concerns the different roles of various users of the system, different modes of use for each, and how the system orients its support to the current user/mode. An influential set of concepts has come from the world of artificial intelligence, such as the "incremental enrichment" style of LISP program development [@Barstow and Shrobe 1981].

3.2.5 APSE Viewed as a Knowledge-Based Expert System

In this view components of an APSE (or conceivably the entire APSE) are created as expert systems based on the past experience of human experts in specific

domains. One issue of IEEE Expert [@IEEE Expert 1986] contains a collection of papers which address this possibility. A paper in this collection [@Zualkernan et al 1986] outlines some steps to determine the feasibility of this approach and treats the specific area of software testing as a case study. This view may be considered a special, advanced case of the preceding view, where the style of user-interaction is that of an "expert assistant."

3.2.6 APSE Viewed as a Stable Framework

This view has been advocated by C.M. McKay of U. Houston Clear Lake [@McKay 1987], and is motivated by the need to support large, complex, non-stop, distributed, long-lived systems such as the NASA Space Station. The APSE is viewed as a stable framework to which new or improved tools can be added over time without interfering with or invalidating previous work. The framework is defined in terms of standard phases and deliverables, in one dimension, and stable interface sets separating tools and various classes of objects, in the other dimension. Ideally, the stability of the framework and its interface sets will allow both the flight system and its support environment to evolve incrementally as reliable, maintainable systems.

3.3 KEY ATTRIBUTES OF WHOLE APSES

This section provides a brief discussion of what appear to be some of the key whole-APSE attributes. The discussion is organized in accordance with the attribute taxonomy [Attributes 6] given in Chapter 6 of this manual, which employs three top-level "acquisition concern" categories:

- Performance Attributes (6.1)
- Design Attributes (6.2)
- Adaptation Attributes (6.3).

Under each of these is a second-level set of "quality factors", such as Efficiency, Integrity, etc. These are further decomposed into "criteria", some of which apply to more than one quality factor. All the criteria attributes are listed alphabetically in Section 6.4 of the Attributes Index.

The definitions given in Chapter 6 are very component-oriented. For example, the definition of Efficiency [Efficiency 6.1.1] includes the words, "The extent to which a component fulfills its purpose using a minimum of computing resources." A whole-APSE version of the definition of Efficiency might read, "The extent to which an APSE supports life cycle phases using a minimum of development-team resources." This same kind of language modification is used throughout the discussion to follow.

The selection of attributes highlighted below has been influenced by discussions within the E&V Team and by the following papers (some of which make reference to many other papers): "Toward Integrated Software Development Environments" [@Wasserman 1981b], "Software Development Environment Issues as Related to Ada" [@Notkin and Haberman 1981], "Essential Properties of IPSEs" [@Lehman and Turski 1987], and "Characteristics and Functions of Software Engineering Environments: An Overview" [@Houghton and Wallace 1987].

A quotation from another paper by Wasserman [@Wasserman 1981a] perhaps best sets the stage for this discussion of key whole-APSE attributes. He says, "The goal is to create an environment that not only enhances developer productivity but also supports the creation of superior products."

3.3.1 Performance Attributes

[Efficiency 6.1.1] - As stated above, efficiency, in the whole-APSE context, is measured in terms of resources used by an entire team in performing an entire phase or major "chunk" of work during development of a software product. This attribute is clearly related to the overall project goal of team productivity.

[Integrity 6.1.2] - This attribute deals with the extent to which access to the software environment or other data is controlled, especially for the purposes of monitoring status, preserving integrity of different versions, and controlling changes. It is an attribute of an important set of management functions that greatly influence team productivity and product quality.

A critical aspect of this attribute concerns the treatment of the unique Ada feature (discussed in Section 3.2.3) associated with inter-unit dependencies. In

managing Ada program library units strict rules apply to the order of compilation. As a project is developed, obsolete units need to be recompiled, compilation order changes, and Ada closure sets may be redefined. Therefore, the APSE's management of Ada source and object modules should be assessed with these special requirements in mind.

[Usability/Anomaly Management 6.1.5/6.4.2]

and

[Usability/Cost 6.1.5/6.4.11]

and

[Usability/Maturity 6.1.5/6.4.18] – These are the aspects of usability that naturally concern managers and controllers of facility investment resources. Anomaly management affects the probability that an APSE will be functionally ready at some *specified point in time. It must operate, of course, on available hardware.* Absence of such availability in a timely way would naturally be disastrous.

Cost includes the cost of purchase or lease, installation, user assistance, and maintenance. Its importance is self-evident.

Maturity is the extent to which an APSE has been used in the development of deliverable software by typical users and to which the feedback from that use has been reflected in improvements.

[Usability/Operability 6.1.5/6.4.20]

and

[Usability/Training 6.1.5/6.4.36] – These represent the human-factors aspects of usability such as ease-of-use, ease-of-learning, on-line help features, and consistency of interfaces. This set of attributes will have a major influence not only on long-term productivity, but on the early acceptance of an APSE by individuals and the team as a whole. The resulting impact on motivation and team spirit can be crucial.

3.3.2 Design Attributes

[Correctness/Completeness 6.2.1/6.4.9] - This is the extent to which an APSE supports the complete set of operations necessary to perform its intended function, which is to provide full support to a development team across an entire product life cycle. This attribute could be interpreted in a minimal way, listing only truly essential (MAPSE) functions necessary for a particular project. Alternatively, it could be interpreted as a list of functions desired to provide the capability to produce high-quality products.

[Maintainability/Self-Descriptiveness 6.2.2/6.4.28] - In the whole-APSE context the aspect of this attribute to be emphasized concerns the underlying data base or schema used to store and retrieve project data. In one of the papers cited above [@Lehman and Turski 1987] this attribute was called data-structuredness. The Ada-Europe document "Selecting an Ada Environment" [@Lyons and Nissen 1986] says, "It is now generally recognized that the totality of information that a project must store, consists not only of individual entities containing data but also of the relationships between them, such as the facts that a particular object file has been compiled from a particular source file, that a source file implements a particular specification, or that an error report relates to a particular release of a system."

[Verifiability, Testability 6.2.3] - This is the extent to which an APSE facilitates evaluation of its own performance, so that productivity and quality metrics can be gathered and analyzed. It is through the use of this attribute that the extensibility (see below) of an APSE can be exploited effectively in a continual process of improvement.

3.3.3 Adaptation Attributes

[Expandability/Augmentability 6.3.1/6.4.4] - This is the extent to which an APSE facilitates the addition of new capabilities in response to needs that go beyond its original requirements. The new capabilities could include new functions, expanded data capacity, or new types of data relationships. The word used to describe this feature in several of the papers cited above is extensibility. Given the current state of the art and rapid pace of change of environment technology, the presence of this

quality (however it is achieved) appears necessary in order to provide a path for evolutionary improvements.

[Interoperability/Commonality 6.3.2/6.4.7] - This is the ability of APSEs to exchange data base objects and their relationships without conversion of formats, and the use of interface standards (e.g., CAIS [@CAIS]) to facilitate such exchanges.

[Transportability 6.3.4] - This is the extent to which an APSE supports the movement of software components to or from another APSE without change in functionality or reprogramming, and the use of interface standards (e.g., CAIS) to facilitate such movements.

3.4 APPROACHES TO WHOLE-APSE E&V

This section provides a brief discussion of approaches to whole-APSE assessment, including both evaluation of performance and quality, and validation of conformance to standards. References to Guidebook sections and other documents are included, in cases where there are known examples of existing assessors or assessment products under development. Such assessors can be categorized generally as either "tools" or "aids." A more refined breakdown is the following:

- Benchmarks and Test Suites (Tools)
- Questionnaires (Aids)
- Monitored Experiments (Aids that may use Tools)
- Decision Aids (Aids that may use Tools and other Aids).

These are discussed, in turn, in the following four subsections.

3.4.1 Benchmarks and Test Suites

Benchmarks are standard tests used to measure the execution, performance or acceptability of an APSE function or set of functions. A test suite is an organized collection of such tests. The Ada Compiler Validation Capability

[@ACVC 1986] is a test suite designed to test conformance of an Ada compiler to the formal definition of the language [@DoD 1983]. A prototype compiler performance evaluation test suite has been generated by the Institute for Defense Analysis [@GB: 6.1], and a more carefully engineered set known as the ACEC or Ada Compiler Evaluation Capability [@GB: 6.2] is being developed, under an E&V Task contract, by the Boeing Company. Another collection of Ada compiler performance tests has been gathered by the ACM SIGAda Performance Issues Working Group [@PIWG 1987]. Results of these tests, run on a number of commercial compilers, will be published in the open literature.

3.4.2 Questionnaires

Questionnaires are used to gather data from vendors or users of tools and APSEs. Examples of such data might include specification parameters, design features, historical information, typical usage scenarios, implementation strategies, enhancement plans or desires, and problem reports. An early example, in Ada terms, of such a questionnaire was one applied to the evaluation of the ROLM Ada Work Center [@Castor 1983].

A whole-APSE-oriented example of the use of questionnaires is the book "Selecting an Ada Environment" [@Lyons and Nissen 1986], which is synopsized in the E&V Guidebook [@GB: 4.12]. The book is organized around background discussions of various topics followed by appropriate questions addressing each topic. The final chapter begins with the following set of high-level questions appropriate for a potential purchaser of an APSE:

a) What does the environment consist of?

b) In particular, what tools are supplied?

c) What are the deliverables?

d) What does it cost?

e) What support is available?

f) Can extra tools be added to the environment easily?

g) What are the conditions of use (number of users, involvement of third parties, etc)?

h) What hardware and software resources (including licenses) are needed to support the environment?"

More detailed questions follow, under specific headings such as support, interfaces, and other issues.

3.4.3 Monitored Experiments

Monitored experiments, based on model projects involving an aggregation of APSE functions or tools, can be performed using APSEs or APSE components to gather data in a systematic and controlled manner. These experiments can be used for both qualitative and quantitative assessments of the functionality, usability, and performance, as well as for the more informal characteristics of APSEs.

3.4.4 Decision Aids

Decision aids allow a user to assess an APSE from a particular point of view. Decision aids may combine the results of a number of tests, questionnaires, and/or monitored experiments, each of which is weighted according to its value for the view being considered [@GB 3].

4. LIFE CYCLE PHASES

This chapter deals with the life cycle phases served by the support environment. The foundation for this discussion is DoD-STD-2167 [@DoD-STD-2167]. The phases that are chosen to represent the progression of activities depend on the view that a person has of the APSE. If one thinks of the APSE as only providing an environment for developing software, then the phases can be limited to the following six:

- Software Requirements Analysis
- Preliminary Design
- Detailed Design
- Coding and Unit Testing
- CSC Integration and Testing
- CSCI Testing.

However, if one sees the APSE as providing support across the entire system life cycle (Integrated Project Support Environment, or IPSE, rather than APSE) [Whole APSE Issues 3.], then the following five phases should be added as well:

- System Concepts
- System Requirements Analysis
- System Integration and Testing
- Operational Testing and Evaluation
- Change Requirements.

Of the above five phases, the first two precede the six software development phases and the next two follow the six phases in the overall system life cycle. The last phase, change requirements, is a phase that may be started in parallel with any of the other phases and addresses the issues of enhancement, error correction, and modification.

Also, a global life cycle 'phase' is introduced. This is not a true life cycle phase, but is really a phase-less abstraction. This 'phase' provides a convenient way to show that certain functions perform services across the entire life cycle that are not specific to a single phase or group of phases. An example of a global function is general purpose text editing.

Chapter 5 of DoD-STD-2167 lists the detailed activities for each of the six software development life cycle phases. These activities fall into the following four general areas:

- Management
- Transformation
- Analysis
- Operation and Support.

These four activities are used as a second-level of classification under each top-level life cycle phase division of this chapter.

Figure 4-1 shows the relationships between the life cycle phases and the other elements in the Reference Manual. Each of the life cycle phases are described in the following sections and under each section are four subsections that deal with each of the general areas of activities. The functions specified for each of the life cycle phases are those that are typically used in the phase.

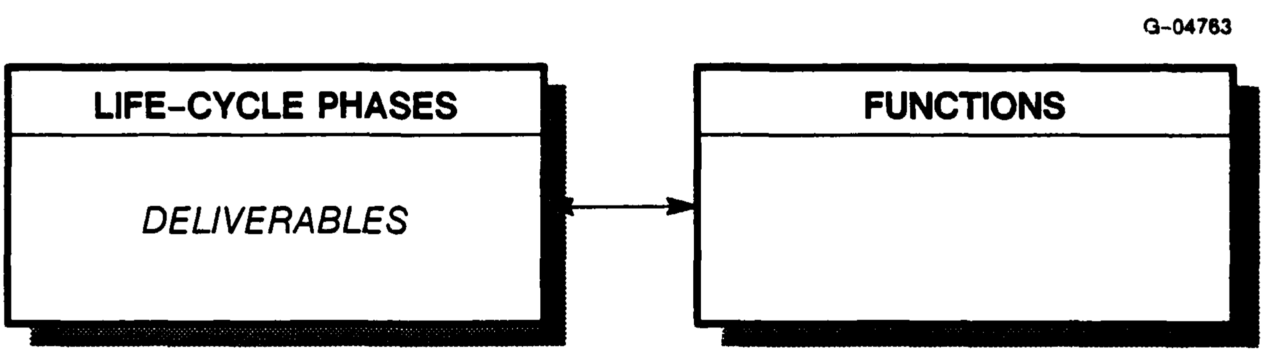


Figure 4-1 Life Cycle Phase Relationships

4.1 SYSTEM CONCEPTS

Description:

4.1.1 Management

Cross References:

Deliverables:

Functions:

4.1.2 Transformation

Cross References:

Deliverables:

Functions:
[Text Editing 7.1.1.1]

4.1.3 Analysis

Cross References:

Deliverables:

Functions:

4.1.4 Operation And Support

Cross References:

Deliverables:

Functions:

4.2 SYSTEM REQUIREMENTS ANALYSIS

Description:

4.2.1 Management

Cross References:

Deliverables:

Functions:

4.2.2 Transformation

Cross References:

Deliverables:

[Operational Concept Document	(OCD),
System/Segment Specification	(SSS),
Prime Item Development Specification]	

Functions:

[Text Editing	7.1.1.1,
Predefined and User-Defined Forms	7.1.2.3,
System Requirements Translation	7.1.6.1,
Requirements to Natural Language Translation	7.1.6.3]

4.2.3 Analysis

Cross References:

Deliverables:

Functions:

[Requirements Simulation	7.3.2.1,
Requirements Prototyping	7.3.2.2]

4.2.4 Operation And Support

Cross References:

Deliverables:

Functions:

4.3 SOFTWARE REQUIREMENTS ANALYSIS

Description:

Those activities of the life cycle pertaining to the establishment of system requirements and a complete set of functional, performance, and interface requirements for each CSCI. [@DoD-STD-2167: 5.1]

4.3.1 Management

Cross References:

Deliverables:

[Software Development Plan	(SDP),
Software Standards and Procedures Manual	(SSPM),
Software Configuration Management Plan	(SCMP),
Software Quality Evaluation Plan	(SQEP)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Specification Management	7.2.1.6,
Program Library Management	7.2.1.7,
Resource Estimation	7.2.2.5]

4.3.2 Transformation

Cross References:

Deliverables:	
[Operational Concept Document	(OCD),
Software Requirements Specification	(SRS),
Interface Requirements Specification	(IRS)]
Functions:	
[Text Editing	7.1.1.1,
Predefined and User-Defined Forms	7.1.2.3,
Software Requirements Translation	7.1.6.2,
Requirements to Natural Language Translation	7.1.6.3]

4.3.3 Analysis

Cross References:

Deliverables:

Functions:

[Tracking	7.2.2.6,
Data Flow Analysis	7.3.1.3,
Functional Analysis	7.3.1.4,
Interface Analysis	7.3.1.5,
Traceability Analysis	7.3.1.6,
Testability Analysis	7.3.1.7,
Test Condition Analysis	7.3.1.8,
Quality Analysis	7.3.1.9,
Requirements Simulation	7.3.2.1,
Requirements Prototyping	7.3.2.2]

4.3.4 Operation And Support

Cross References:

Deliverables:

Functions:

4.4 PRELIMINARY DESIGN

Description:

Those activities in the life cycle pertaining to the development of a modular; top-level design of each CSCI from the software requirements. [@DoD-STD-2167: 5.2]

4.4.1 Management

Cross References:

Deliverables:

[Software Development Plan	(SDP),
Software Standards and Procedures Manual	(SSPM),
Software Configuration Management Plan	(SCMP),
Software Quality Evaluation Plan	(SQEP)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Specification Management	7.2.1.6,
Program Library Management	7.2.1.7,
Resource Estimation	7.2.2.5]

4.4.2 Transformation

Cross References:

Deliverables:

[Software Top Level Design Document	(STLDD),
Software Detailed Design Document	(SDDD),
Interface Design Document	(IDD),
Data Base Design Document	(DBDD)]

Functions:

[Text Editing	7.1.1.1,
Predefined and User-Defined Forms	7.1.2.3,
Preliminary Design Translation	7.1.6.4,
Design Generation	7.1.7.1,
Requirements Reconstruction	7.1.7.2]

4.4.3 Analysis

Cross References:

Deliverables:

[Software Test Plan	(STP)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Tracking	7.2.2.6,
Interface Analysis	7.3.1.5,
Quality Analysis	7.3.1.9,
Complexity Measurement	7.3.1.10,
Completeness Checking	7.3.1.12,
Consistency Checking	7.3.1.13,
Cross Reference	7.3.1.17,
Invocation Analysis	7.3.1.19,
Scanning	7.3.1.20,
Structured Walkthrough	7.3.1.21
Simulation and Modeling	7.3.2.3,
Design Prototyping	7.3.2.4,
Formal Verification	7.3.3]

4.4.4 Operation And Support

Cross References:

Deliverables:

[Computer System Operator's Manual	(CSOM),
Software User's Manual	(SUM),
Computer System Diagnostic Manual	(CSDM),
Computer Resources Integrated Support Document	(CRISD)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3]

4.5 DETAILED DESIGN

Description:

Those activities in the life cycle pertaining to the development of a modular, detailed design of each CSCI from the preliminary design. [@DoD-STD-2167: 5.3]

4.5.1 Management

Cross References:

Deliverables:

[Software Development Plan	(SDP),
Software Standards and Procedures Manual	(SSPM),
Software Configuration Management Plan	(SCMP),
Software Quality Evaluation Plan	(SQEP)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Specification Management	7.2.1.6,
Program Library Management	7.2.1.7,
Resource Estimation	7.2.2.5]

4.5.2 Transformation

Cross References:

Deliverables:	
[Software Detailed Design Document	(SDDD),
Interface Design Document	(IDD),
Data Base Design Document	(DBDD),
Software Development File	(SDF)]
Functions:	
[Text Editing	7.1.1.1,
Predefined and User-Defined Forms	7.1.2.3,
Detailed Design Translation	7.1.6.5,
Design Generation	7.1.7.1,
Requirements Reconstruction	7.1.7.2]

4.5.3 Analysis

Cross References:

Deliverables:	
[Software Test Description	(STD)]
Functions:	
[Predefined and User-Defined Forms	7.1.2.3,
Tracking	7.2.2.6,
Interface Analysis	7.3.1.5,
Quality Analysis	7.3.1.9,
Complexity Measurement	7.3.1.10,
Completeness Checking	7.3.1.12,
Consistency Checking	7.3.1.13,
Cross Reference	7.3.1.17,
Invocation Analysis	7.3.1.19,
Scanning	7.3.1.20,
Structured Walkthrough	7.3.1.21,
Simulation and Modeling	7.3.2.3,
Design Prototyping	7.3.2.4,
Formal Verification	7.3.3,
Symbolic Execution	7.3.4]

4.5.4 Operation And Support

Cross References:

Deliverables:

[Computer System Operator's Manual	(CSOM),
Software User's Manual	(SUM),
Computer System Diagnostic Manual	(CSDM),
Computer Resources Integrated Support Document	(CRISD),
Software Programmer's Manual	(SPM),
Firmware Support Manual	(FSM)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3]

4.6 CODING AND UNIT TESTING

Description:

Those activities in the life cycle pertaining to the coding and testing of each unit comprising the detailed design. [@DoD-STD-2167: 5.4]

4.6.1 Management

Cross References:

Deliverables:

[Software Development Plan	(SDP),
Software Standards and Procedures Manual	(SSPM),
Software Configuration Management Plan	(SCMP),
Software Quality Evaluation Plan	(SQEP)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Program Library Management	7.2.1.7,
Test Data Management	7.2.1.8,
Resource Estimation	7.2.2.5]

4.6.2 Transformation

Cross References:

Deliverables:	
[Software Development File	(SDF)]
Functions:	
[Text Editing	7.1.1.1,
Predefined and User-Defined Forms	7.1.2.3,
Assembling	7.1.6.6,
Compilation	7.1.6.7,
Conversion	7.1.6.8,
Macro Expansion	7.1.6.9,
Structure Preprocessing	7.1.6.10,
Body Stub Generation	7.1.6.11,
Preamble Generation	7.1.6.12,
Linking/Loading	7.1.6.13,
Interpretation	7.1.6.14,
Requirements Reconstruction	7.1.7.2,
Program Generation	7.1.7.3,
Source Reconstruction	7.1.7.4,
Decompilation	7.1.7.5,
Disassembling	7.1.7.6]

4.6.3 Analysis

Cross References:

Deliverables:	
[Software Test Procedure	(STPR)]
Functions:	
[Predefined and User-Defined Forms	7.1.2.3,
Tracking	7.2.2.6,
Data Flow Analysis	7.3.1.3,
Interface Analysis	7.3.1.5,
Quality Analysis	7.3.1.9,
Complexity Measurement	7.3.1.10,
Completeness Checking	7.3.1.12,
Consistency Checking	7.3.1.13,
Cross Reference	7.3.1.17,
Invocation Analysis	7.3.1.19,
Scanning	7.3.1.20,
Structured Walkthrough	7.3.1.21,
Auditing	7.3.1.22,
Error Checking	7.3.1.23,
Statistical Analysis	7.3.1.24,
Statistical Profiling	7.3.1.25,
Structure Checking	7.3.1.26,
Type Analysis	7.3.1.27,
Units Analysis	7.3.1.28,
I/O Specification Analysis	7.3.1.29,
Debugging	7.3.2.5,
Executable Assertion Checking	7.3.2.6,
Constraint Evaluation (Contention)	7.3.2.7,
Coverage/Frequency Analysis	7.3.2.8,
Mutation Analysis	7.3.2.9,
Testing	7.3.2.10,
Regression Testing	7.3.2.11,
Resource Utilization	7.3.2.12,
Emulation	7.3.2.13,
Timing Analysis	7.3.2.14,
Tuning	7.3.2.15,
Formal Verification	7.3.3,
Symbolic Execution	7.3.4]

4.6.4 Operation And Support

Cross References:

Deliverables:

[Computer System Operator's Manual	(CSOM),
Software User's Manual	(SUM),
Computer System Diagnostic Manual	(CSDM)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3]

4.7 CSC INTEGRATION AND TESTING

Description:

Those activities in the life cycle pertaining to the integration of units of code and the performance of informal tests on aggregates of integrated units. [@DoD-STD-2167: 5.5]

4.7.1 Management

Cross References:

Deliverables:

[Software Development Plan	(SDP),
Software Standards and Procedures Manual	(SSPM),
Software Configuration Management Plan	(SCMP),
Software Quality Evaluation Plan	(SQEP)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Program Library Management	7.2.1.7,
Test Data Management	7.2.1.8,
Resource Estimation	7.2.2.5]

4.7.2 Transformation

Cross References:

Deliverables:

[Software Development File	(SDF)]

Functions:

[Text Editing	7.1.1.1,
Predefined and User-Defined Forms	7.1.2.3,
Assembling	7.1.6.6,
Compilation	7.1.6.7,
Conversion	7.1.6.8,
Macro Expansion	7.1.6.9,
Structure Preprocessing	7.1.6.10,
Body Stub Generation	7.1.6.11,
Preamble Generation	7.1.6.12,
Linking/Loading	7.1.6.13,
Interpretation	7.1.6.14,
Requirements Reconstruction	7.1.7.2,
Program Generation	7.1.7.3,
Source Reconstruction	7.1.7.4,
Decompilation	7.1.7.5,
Disassembling	7.1.7.6]

4.7.3 Analysis

Cross References:

Deliverables:

[Software Test Procedure	(STPR)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Tracking	7.2.2.6,
Data Flow Analysis	7.3.1.3,
Interface Analysis	7.3.1.5,
Quality Analysis	7.3.1.9,
Complexity Measurement	7.3.1.10,
Completeness Checking	7.3.1.12,
Consistency Checking	7.3.1.13,
Cross Reference	7.3.1.17,
Invocation Analysis	7.3.1.19,
Scanning	7.3.1.20,
Structured Walkthrough	7.3.1.21,
Auditing	7.3.1.22,
Error Checking	7.3.1.23,
Statistical Profiling	7.3.1.25,
Structure Checking	7.3.1.26,
Type Analysis	7.3.1.27,
Units Analysis	7.3.1.28,
I/O Specification Analysis	7.3.1.29,
Debugging	7.3.2.5,
Executable Assertion Checking	7.3.2.6,
Constraint Evaluation (Contention)	7.3.2.7,
Coverage/Frequency Analysis	7.3.2.8,
Mutation Analysis	7.3.2.9,
Testing	7.3.2.10,
Regression Testing	7.3.2.11,
Resource Utilization	7.3.2.12,
Emulation	7.3.2.13,
Timing Analysis	7.3.2.14,
Tuning	7.3.2.15,
Symbolic Execution	7.3.4,
Problem Report Analysis	7.3.5]

4.7.4 Operation And Support

Cross References:

Deliverables:

[Computer System Operator's Manual	(CSOM),
Software User's Manual	(SUM),
Computer System Diagnostic Manual	(CSDM)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3]

4.8 CSCI TESTING

Description:

Those activities in the life cycle pertaining to the conduct of formal tests on each CSCI and the recording/analysis of test results. [@DoD-STD-2167: 5.6]

4.8.1 Management

Cross References:

Deliverables:

[Software Development Plan	(SDP),
Software Standards and Procedures Manual	(SSPM),
Software Configuration Management Plan	(SCMP),
Software Quality Evaluation Plan	(SQEP)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Program Library Management	7.2.1.7,
Test Data Management	7.2.1.8,
Resource Estimation	7.2.2.5]

4.8.2 Transformation

Cross References:

Deliverables:	
[Software Product Specification	(SPS),
Version Description Document	(VDD),
Software Development File	(SDF)]
Functions:	
[Import/Export	7.2.3.6,
Assembling	7.1.6.6,
Compilation	7.1.6.7,
Conversion	7.1.6.8,
Macro Expansion	7.1.6.9,
Structure Preprocessing	7.1.6.10,
Preamble Generation	7.1.6.12,
Linking/Loading	7.1.6.13,
Interpretation	7.1.6.14]

4.8.3 Analysis

Cross References:

Deliverables:

[Software Test Report	(STR)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Tracking	7.2.2.6,
Quality Analysis	7.3.1.9,
Debugging	7.3.2.5,
Coverage/Frequency Analysis	7.3.2.8,
Testing	7.3.2.10,
Regression Testing	7.3.2.11,
Resource Utilization	7.3.2.12,
Emulation	7.3.2.13,
Timing Analysis	7.3.2.14,
Tuning	7.3.2.15,
Problem Report Analysis	7.3.5]

4.8.4 Operation And Support

Cross References:

Deliverables:	
[Computer System Operator's Manual	(CSOM),
Software User's Manual	(SUM),
Computer System Diagnostic Manual	(CSDM)]
Functions:	
[Predefined and User-Defined Forms	7.1.2.3]

4.9 SYSTEM INTEGRATION AND TESTING

Description:

Those activities in the life cycle pertaining to the successive integration and testing of CSCIs and HWCIs to validate that the complete system is properly integrated and satisfies system requirements.

4.9.1 Management

Cross References:

Deliverables:

Functions:

4.9.2 Transformation

Cross References:

Deliverables:

Functions:

[Import/Export	7.2.3.6,
Assembling	7.1.6.6,
Compilation	7.1.6.7,
Conversion	7.1.6.8,
Macro Expansion	7.1.6.9,
Structure Preprocessing	7.1.6.10,
Preamble Generation	7.1.6.12,
Linking/Loading	7.1.6.13,
Interpretation	7.1.6.14]

4.9.3 Analysis

Cross References:

Deliverables:

Functions:

[Import/Export	7.2.3.6,
Problem Report Analysis	7.3.5]

4.9.4 Operation And Support

Cross References:

Deliverables:

Functions:

4.10 OPERATIONAL TESTING AND EVALUATION

Description:

Those activities in the life cycle where the system is tested and evaluated in surroundings which are as operationally realistic as possible to determine that the system will satisfactorily perform the mission for which it was designed.

4.10.1 Management

Cross References:

Deliverables:

Functions:
[Evaluation Results Management 7.2.1.9]

4.10.2 Transformation

Cross References:

Deliverables:

Functions:

[Assembling	7.1.6.6,
Compilation	7.1.6.7,
Conversion	7.1.6.8,
Macro Expansion	7.1.6.9,
Structure Preprocessing	7.1.6.10,
Preamble Generation	7.1.6.12,
Linking/Loading	7.1.6.13,
Interpretation	7.1.6.14]

4.10.3 Analysis

Cross References:

Deliverables:

Functions:

[Import/Export	7.2.3.6,
Problem Report Analysis	7.3.5]

4.10.4 Operation And Support

Cross References:

Deliverables:

Functions:

4.11 CHANGE REQUIREMENTS

Description:

Those activities in the life cycle pertaining to the support (error detection/correction) and enhancement of the operational CSCIs. Often, this activity will result in a series of software developments potentially requiring tool features (besides Global features) different from all preceding life cycle activities.

4.11.1 Management

Cross References:

Deliverables:

Functions:

4.11.2 Transformation

Cross References:

Deliverables:

[Engineering Change Proposal	(ECP),
Specification Change Notice	(SCN)]

Functions:

[Predefined and User-Defined Forms	7.1.2.3,
Assembling	7.1.6.6,
Compilation	7.1.6.7,
Conversion	7.1.6.8,
Macro Expansion	7.1.6.9,
Structure Preprocessing	7.1.6.10,
Preamble Generation	7.1.6.12,
Linking/Loading	7.1.6.13,
Interpretation	7.1.6.14]

4.11.3 Analysis

Cross References:

Deliverables:

Functions:
[Change Impact Analysis 7.3.6.1]

4.11.4 Operation And Support

Cross References:

Deliverables:

Functions:

4.12 GLOBAL

Description:

Represents those general support features which include common (in implementation and use) software elements or functions. Global features can be included in each life cycle activity.

4.12.1 Management

Cross References:

Deliverables:

Functions:

[Data Base (Object) Management	7.2.1.1,
Documentation Management	7.2.1.2,
File Management	7.2.1.3,
Electronic Mail	7.2.1.4,
Electronic Conferencing	7.2.1.5,
Cost Estimation	7.2.2.1,
Quality Specification	7.2.2.2,
Scheduling	7.2.2.3,
Work Breakdown Structure	7.2.2.4,
Configuration Management	7.2.2.7,
Command Language Processing	7.2.3.1,
Input/Output Support	7.2.3.2,
Kernel	7.2.3.3]

4.12.2 Transformation

Cross References:

Deliverables:

Functions:

[Text Editing	7.1.1.1,
Graphics Editing	7.1.1.3,
MIL-STD Format	7.1.2.1,
Table of Contents	7.1.2.2,
On-Line Assistance Processing	7.1.3,
Sort/Merge	7.1.4,
Graphics Generation	7.1.5,
Runtime Environment	7.2.3.5]

4.12.3 Analysis

Cross References:

Deliverables:

Functions:

[Math/Statistics	7.2.3.6,
Comparison	7.3.1.1,
Spelling Checking	7.3.1.2]

4.12.4 Operation And Support

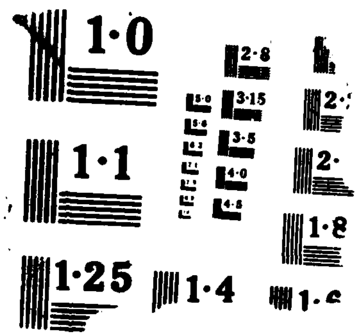
Cross References:

Deliverables:

Functions:

AD-A193 309 E&V (EVALUATION AND VALIDATION) MANUAL VERSION 10(U) 2/4
ADA JOINT PROGRAM OFFICE ARLINGTON VA 29 DEC 87
TASC-TR-5234-3

UNCLASSIFIED F/G 12/5 NL


1·0
2·8
3·15
3·5
1·1
4·0
4·5
1·8
1·25
1·4

5. APSE TOOL CATEGORIES

This chapter deals with APSEs, toolsets, and tools. E&V technology is always applied to APSEs or their components and thus this index will be a natural starting point for many users of the RM. The following sections describe the APSE and its components and relate the components to functions as shown in Fig. 5-1. The relationships between tools and functions given in this chapter are typical (or traditional) relationships. The real capabilities should be determined by the tool specifications, marketing claims, or the like.

There is also a relationship between tools and attributes. Some attributes are related to qualities of the function(s) performed (such as completeness) by the software, while other attributes relate to non-functional aspects (such as modularity) of the software. Thus, to determine how to assess a tool, both the tool-function relationships and the attributes must be explored to identify the metrics to be used.

APSEs are the highest level of toolset. APSEs should contain all the tools and toolsets needed to support the full spectrum of project activities. APSEs, like

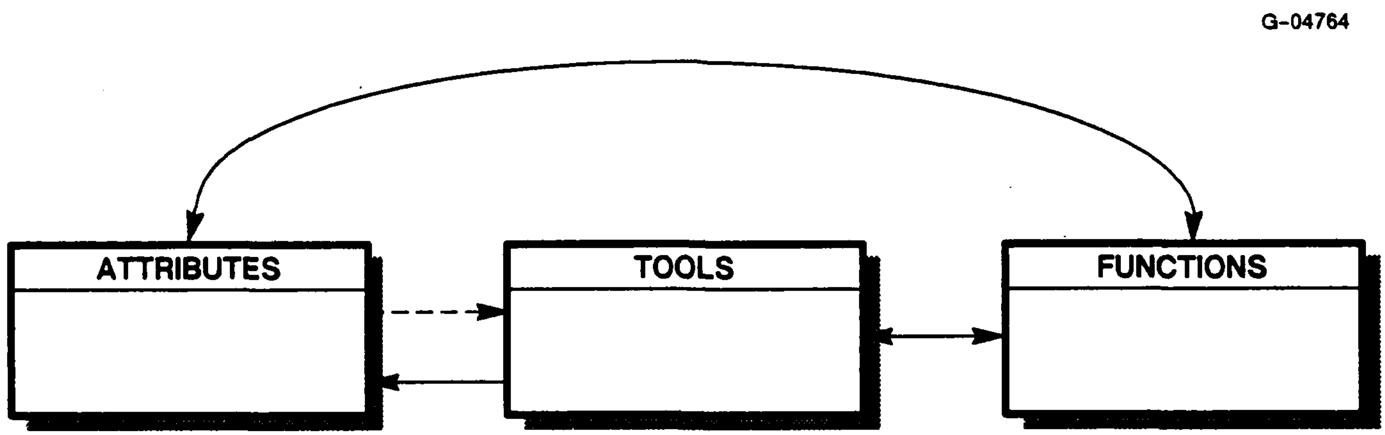


Figure 5-1 Tool Relationships

toolsets, have characteristics that may be assessed as a whole [Whole APSE Issues 3.1]. Toolsets are one or more tools which are intimately coupled and support a related set of functions within the APSE. Tools are the smallest APSE components which can be independently acquired. Each section in this chapter describes a toolset category that might be found in an APSE. The subsections within each section give the tool categories that comprise the toolset. Not all toolsets are necessary for an APSE; likewise, for a specific toolset, all tools may not always be present.

5.1 COMPUTER MANAGEMENT SYSTEM

Description:

Those tools needed to access, use, and maintain the hardware and software which consists of the APSE and the system on which it runs.

5.1.1 Command Language Processor

Cross References:

Functions:
[Command Language Processing 7.2.3.1]

5.1.2 Archive, Backup, and Retrieval System

Cross References:

Functions:
[Import/Export 7.2.3.6]

5.1.3 Security System

Cross References:

Functions:
[Kernel 7.2.3.3]

5.1.4 Job Scheduler

Cross References:

Functions:
[Runtime Environment 7.2.3.5]

5.1.5 Resource Controller

Cross References:

Functions:
[Runtime Environment 7.2.3.5]

5.1.6 File Manager

Cross References:

Functions:
[File Management 7.2.1.3]

5.1.7 Import/Export System

Cross References:

Functions:
[Import/Export 7.2.3.6]

5.1.8 On-Line Assistance Processor

Cross References:

Functions:
[On-Line Assistance Processing 7.1.3]

5.1.9 Data Base Manager

Cross References:

Functions:
[Data Base Management 7.2.1.1]

5.2 PROJECT MANAGEMENT SYSTEM

Description:

Those tools needed to plan, develop, and maintain an applications system.

5.2.1 Cost Estimator

Cross References:

Functions:
[Cost Estimation 7.2.2.1]

5.2.2 Quality Analyzer

Cross References:

Functions:
[Quality Specification 7.2.2.2,
Quality Assessment 7.2.2.8]

5.2.3 Scheduler

Cross References:

Functions:
[Scheduling 7.2.2.3]

5.2.4 Work Breakdown Structure

Cross References:

Functions:
[Work Breakdown Structure 7.2.2.4]

5.2.5 Resource Estimator

Cross References:

Functions:
[Resource Estimation 7.2.2.5]

5.2.6 Tracking

Cross References:

Functions:
[Tracking 7.2.2.6]

5.2.7 Configuration Manager

Cross References:

Functions:
[Configuration Management 7.2.2.7]

5.2.8 Problem Report Analyzer

Cross References:

Functions:
[Problem Report Analysis 7.3.5]

5.2.9 Change Request Analyzer

Cross References:

Functions:
[Change Request Analysis 7.3.6]

5.3 COMPILATION SYSTEM

Description:

Those tools needed to produce and run software systems.

5.3.1 Program Library Manager

Cross References:

Functions:

[Program Library Management	7.2.1.7]

5.3.2 Syntax-Directed Editor

Cross References:

Functions:

[Text Editing	7.1.1.1,
Syntax And Semantics Checking	7.3.1.15]

5.3.3 Compiler

Cross References:

Functions:

[Compilation	7.1.6.7]

5.3.4 Assembler

Cross References:

Functions:
[Assembling 7.1.6.6]

5.3.5 Linker

Cross References:

Functions:
[Linking/Loading 7.1.6.13]

5.3.6 Loader

Cross References:

Functions:
[Linking/Loading 7.1.6.13]

5.3.7 Interpreter

Cross References:

Functions:
[Interpretation 7.1.6.14]

5.3.8 Runtime Library

Cross References:

Functions:

[Input/Output Support	7.2.3.2,
Math/Statistics	7.2.3.4,
Runtime Environment	7.2.3.5]

5.4 DOCUMENT SYSTEM

Description:

Those tools needed to develop and produce documents.

5.4.1 Document Manager

Cross References:

Functions:

[Document Management	7.2.1.2]

5.4.2 Word Processor

Cross References:

Functions:

[Text Editing	7.1.1.1,
Syntax And Semantics Checking	7.3.1.15]

5.4.3 Spell Checker

Cross References:

Functions:

[Spelling Checking	7.3.1.2]

5.4.4 Graphics Generator

Cross References:

Functions:
[Graphics Generation 7.1.5]

5.4.5 Formatter

Cross References:

Functions:
[Formatting 7.1.2]

5.5 DESKTOP SYSTEM

Description:

Those tools needed to do the every-day, administrative tasks of business.

5.5.1 Spreadsheet

Cross References:

Functions:	
[Formatting	7.1.2,
Math/Statistics	7.2.3.4]

5.5.2 Calculator

Cross References:

Functions:	
[Math/Statistics	7.2.3.4]

5.5.3 Address Book

Cross References:

Functions:	
[Sort/Merge	7.1.4]

5.5.4 Electronic Mail

Cross References:

Functions:
[Electronic Mail 7.2.1.4]

5.5.5 Phone Book

Cross References:

Functions:
[Sort/Merge 7.1.4]

5.5.6 Electronic Conferencing

Cross References:

Functions:
[Electronic Conferencing 7.2.1.5]

5.5.7 Calendar

Cross References:

Functions:
[Scheduling 7.2.2.3]

5.5.8 Dictionary

Cross References:

Functions:
[Sort/Merge 7.1.4]

5.6 STATIC ANALYZER SYSTEM

Description:

Those tools needed to do analysis of software systems without actually executing the program(s) being analyzed.

5.6.1 Comparator

Cross References:

Functions:
[Comparison 7.3.1.1]

5.6.2 Data Flow Analyzer

Cross References:

Functions:
[Data Flow Analysis 7.3.1.3]

5.6.3 Functional Analyzer

Cross References:

Functions:
[Functional Analysis 7.3.1.4]

5.6.4 Interface Analyzer

Cross References:

Functions:
[Interface Analysis 7.3.1.5]

5.6.5 Traceability Analyzer

Cross References:

Functions:
[Traceability Analysis 7.3.1.6]

5.6.6 Testability Analyzer

Cross References:

Functions:
[Testability Analysis 7.3.1.7]

5.6.7 Test Condition Analyzer

Cross References:

Functions:
[Test Condition Analysis 7.3.1.8]

5.6.8 Quality Analyzer

Cross References:

Functions:
[Quality Analysis 7.3.1.9]

5.6.9 Complexity Measurer

Cross References:

Functions:
[Complexity Measurement 7.3.1.10]

5.6.10 Correctness Checker

Cross References:

Functions:
[Correctness Checking 7.3.1.11]

5.6.11 Completeness Checker

Cross References:

Functions:
[Completeness Checking 7.3.1.12]

5.6.12 Consistency Checker

Cross References:

Functions:
[Consistency Checking 7.3.1.13]

5.6.13 Reusability Analyzer

Cross References:

Functions:
[Reusability Analysis 7.3.1.14]

5.6.14 Syntax And Semantics Checker

Cross References:

Functions:
[Syntax And Semantics Checking 7.3.1.15]

5.6.15 Reachability Analyzer

Cross References:

Functions:
[Reachability Analysis 7.3.1.16]

5.6.16 Cross Referencer

Cross References:

Functions:
[Cross Reference 7.3.1.17]

5.6.17 Maintainability Analyzer

Cross References:

Functions:
[Maintainability Analysis 7.3.1.18]

5.6.18 Invocation Analyzer

Cross References:

Functions:
[Invocation Analysis 7.3.1.19]

5.6.19 Scanner

Cross References:

Functions:
[Scanning 7.3.1.20]

5.6.20 Structured Walkthrough Tool

Cross References:

Functions:
[Structured Walkthrough 7.3.1.21]

5.6.21 Auditor

Cross References:

Functions:
[Auditing 7.3.1.22]

5.6.22 Error Checker

Cross References:

Functions:
[Error Checking 7.3.1.23]

5.6.23 Statistical Analyzer

Cross References:

Functions:
[Statistical Analysis 7.3.1.24]

5.6.24 Statistical Profiler

Cross References:

Functions:
[Statistical Profiling 7.3.1.25]

5.6.25 Structure Checker

Cross References:

Functions:
[Structure Checking 7.3.1.26]

5.6.26 Type Analyzer

Cross References:

Functions:
[Type Analysis 7.3.1.27]

5.6.27 Units Analyzer

Cross References:

Functions:
[Units Analysis 7.3.1.28]

5.6.28 I/O Specification Analyzer

Cross References:

Functions:
[I/O Specification Analysis 7.3.1.29]

5.6.29 Sizing Analyzer

Cross References:

Functions:
[Sizing Analysis 7.3.1.30]

5.7 DYNAMIC ANALYZER SYSTEM

Description:

Those tools needed to do analysis of software systems while actually executing the program(s) or some representation of the system being analyzed.

5.7.1 Requirements Simulator

Cross References:

Functions:
[Requirements Simulation 7.3.2.1]

5.7.2 Requirements Prototype

Cross References:

Functions:
[Requirements Prototyping 7.3.2.2]

5.7.3 Simulation And Modelling Tools

Cross References:

Functions:
[Simulation And Modelling 7.3.2.3]

5.7.4 Design Prototype

Cross References:

Functions:
[Design Prototyping 7.3.2.4]

5.7.5 Debugger

Cross References:

Functions:
[Debugging 7.3.2.5]

5.7.6 Executable Assertion Checker

Cross References:

Functions:
[Executable Assertion Checking 7.3.2.6]

5.7.7 Constraint Evaluator

Cross References:

Functions:
[Constraint Evaluation 7.3.2.7]

5.7.8 Coverage/Frequency Analyzer

Cross References:

Functions:
[Coverage/Frequency Analysis 7.3.2.8]

5.7.9 Mutation Analyzer

Cross References:

Functions:
[Mutation Analysis 7.3.2.9]

5.7.10 Testing Analyzer

Cross References:

Functions:
[Testing 7.3.2.10]

5.7.11 Regression Testing Analyzer

Cross References:

Functions:
[Regression Testing 7.3.2.11]

5.7.12 Resource Utilization Analyzer

Cross References:

Functions:
[Resource Utilization 7.3.2.12]

5.7.13 Emulator

Cross References:

Functions:
[Emulation 7.3.2.13]

5.7.14 Timing Analyzer

Cross References:

Functions:
[Timing Analysis 7.3.2.14]

5.7.15 Tuning Analyzer

Cross References:

Functions:
[Tuning 7.3.2.15]

5.7.16 Reliability Analyzer

Cross References:

Functions:
[Reliability Analysis 7.3.2.16]

5.7.17 Real Time Analyzer

Cross References:

Functions:
[Real Time Analysis 7.3.2.17]

5.7.18 Formal Verification System

Cross References:

Functions:
[Formal Verification 7.3.3]

5.7.19 Symbolic Execution System

Cross References:

Functions:
[Symbolic Execution 7.3.4]

THIS PAGE LEFT INTENTIONALLY BLANK

6. ATTRIBUTES

Attributes are the characteristics of tools or "whole APSEs" which the E&V user evaluates (or validates) to make assessments and comparisons. Therefore, the attributes are integral to finding E&V technology since all E&V technology is used to assess tools or APSEs for one or more attributes. This manual uses a hierarchy of software attributes derived from one presented in a report by the Rome Air Development Center [@RADC 1985]. The focus of the report is planning and designing quality into application software throughout the software life cycle. Nevertheless, the report provides an excellent framework for understanding all the issues that surround software quality and is recommended for those users seeking more details into this complex process. The RADC report is therefore chosen as a basis for this section of the E&V Reference Manual even though the focus here is on system and support software and also on specifying and assessing attributes for software that is already developed. The remainder of this introductory section is used to summarize some of the issues that must be dealt with in understanding the software attributes. These are:

- The attribute hierarchy
- The functional dependence of attributes
- Guidance in the selection of attributes
- Attribute interrelationships.

The first two levels of the attribute hierarchy are shown in Table 6-1. The highest level of the hierarchy (performance, design, and adaptation) show three broad categories of acquisition concerns to potential users with regard to software. The next level shows quality factors which are user-oriented terms, each representing an aspect of software (or tool) quality. The quality factors can themselves be decomposed into various criteria as shown in Table 6-2. The criteria are software-oriented terms representing software characteristics. The degree to which these characteristics are

TABLE 6-1
TOP LEVEL ATTRIBUTE HIERARCHY

G-04769

ACQUISITION CONCERN	USER CONCERN	QUALITY FACTOR
PERFORMANCE- HOW WELL DOES IT FUNCTION?	HOW WELL DOES IT UTILIZE A RESOURCE?	EFFICIENCY
	HOW SECURE IS IT?	INTEGRITY
	WHAT CONFIDENCE CAN BE PLACED IN WHAT IT DOES?	RELIABILITY
	HOW WELL WILL IT PERFORM UNDER ADVERSE CONDITIONS?	SURVIVABILITY
	HOW EASY IS IT TO USE?	USABILITY
DESIGN- HOW WELL IS IT DESIGNED?	HOW WELL DOES IT CONFORM TO THE REQUIREMENTS?	CORRECTNESS
	HOW EASY IS IT TO REPAIR?	MAINTAINABILITY
	HOW EASY IS IT TO VERIFY ITS PERFORMANCE?	VERIFIABILITY, TESTABILITY
ADAPTATION- HOW ADAPTABLE IS IT?	HOW EASY IS IT TO EXPAND, CHANGE, OR UPGRADE ITS CAPABILITY OR PERFORMANCE?	EXPANDABILITY, FLEXIBILITY
	HOW EASY IS IT TO INTERFACE WITH ANOTHER SYSTEM?	INTEROPERABILITY
	HOW EASY IS IT TO CONVERT FOR USE IN ANOTHER APPLICATION?	REUSABILITY
	HOW EASY IS IT TO TRANSPORT?	TRANSPORTABILITY

present in the software is an indication of the degree of presence of a quality factor. The criteria can support more than one of the software quality factors at the next higher level.

The quality criteria can also be decomposed into metrics which are software-oriented details of the software characteristics. These metrics represent specific questions, checklists, or other tests that are used to determine the extent to which the tool has that characteristic. The metrics must often be tailored to the particular function(s) that the software performs. Other metrics do not deal with functionality, but with aspects of the software that are independent of the function(s)

TABLE 6-2
COMPLETE ATTRIBUTE HIERARCHY

G-04766

ACQUISITION CONCERN	CRITERION/ SECTION		PERFORMANCE 6.1					DESIGN 6.2			ADAPTATION 6.3			
			1 EFFICIENCY	2 INTEGRITY	3 RELIABILITY	4 SURVIVABILITY	5 USABILITY	1 CORRECTNESS	2 MAINTAINABILITY	3 VERIFIABILITY	1 EXPANDABILITY	2 INTEROPERABILITY	3 REUSABILITY	4 TRANSPORTABILITY
PERFORMANCE	ACCURACY	6.4.1			X									
	ANOMALY MANAGEMENT (FAULT OR ERROR TOLERANCE, ROBUSTNESS	6.4.2			X	X								
	AUTONOMY	6.4.5				X								
	CAPACITY	6.4.6					X							
	COMMUNICATION EFFECTIVENESS	6.4.8	X											
	COST	6.4.11					X							
	DISTRIBUTEDNESS	6.4.12				X								
	MATURITY	6.4.18					X							
	OPERABILITY (COMMUNICATIVENESS)	6.4.20					X							
	POWER	6.4.21					X							
	PROCESSING (EXECUTION) EFFECTIVENESS	6.4.22	X											
	RECONFIGURABILITY	6.4.24				X								
	REQUIRED CONFIGURATION	6.4.26					X							
	STORAGE EFFECTIVENESS	6.4.31	X											
	SYSTEM ACCESSIBILTY	6.4.32		X										
	TRAINING	6.4.36					X							
DESIGN	COMPLETENESS	6.4.9						X						
	CONSISTENCY	6.4.10						X	X					
	TRACEABILITY	6.4.35						X						
	VISIBILITY (TEST AVAILABILITY)	6.4.38							X	X				
ADAPTATION	APPLICATION INDEPENDENCE	6.4.3											X	
	AUGMENTABILITY	6.4.4									X			
	COMMONALITY (DATA AND COMMUNICATION)	6.4.7										X		
	DOCUMENT ACCESSIBILITY	6.4.13											X	
	FUNCTIONAL OVERLAP	6.4.14										X		
	FUNCTIONAL SCOPE	6.4.15											X	
	GENERALITY	6.4.16									X		X	
	REHOSTABILITY (INDEPENDENCE)	6.4.25										X	X	X
	RETARGETABILITY (INDEPENDENCE)	6.4.27									X	X	X	X
	SYSTEM CLARITY	6.4.33											X	
	SYSTEM COMPATIBILITY	6.4.34										X		
	VIRTUALITY	6.4.37									X			
GENERAL	GRANULARITY	6.4.17				X	X		X	X	X		X	
	MODULARITY	6.4.19				X			X	X	X	X	X	X
	PROPRIETARY RIGHTS	6.4.23							X		X			
	SELF-DESCRIPTIVENESS	6.4.28							X	X	X		X	X
	SIMPLICITY	6.4.29			X				X	X	X		X	
	SOFTWARE PRODUCTION VEHICLE (S)	6.4.30							X		X			

performed. Because of the possible functional dependence of the software quality metrics, the metrics are not listed in this manual, but are found in the Guidebook [@GB].

The RADC report not only defines the software quality factors, criteria, and metrics, it also provides guidance for their selection and use. In particular, there is guidance for selecting attributes based on:

- System characteristics
- Complementary factors
- Shared criteria
- Attribute interrelationships.

The desired software quality factors should be tailored to the characteristics of the software being built and the development environment. In fact, some of the tools selected for the development environment will probably be based on the system characteristics. For example, if a tool or environment is to have a long life cycle, the E&V user should be concerned with expandability and maintainability. Likewise, if the system being built is a real-time application, the developer should consider acquiring tools that can assess the efficiency of the system. Table 6-3 lists some application and environment characteristics and the related software quality factors that are likely to be important.

Another consideration when selecting quality factors to be assessed is the effect of low quality levels among complementary factors. Four factors are complementary to most other factors and should influence the selection of important factors. Table 6-4 shows the complementary factors:

- Reliability
- Correctness
- Maintainability
- Verifiability.

TABLE 6-3

EXAMPLES OF APPLICATION/ENVIRONMENT CHARACTERISTICS AND RELATED SOFTWARE QUALITY FACTORS

G-04765

APPLICATION/ENVIRONMENT CHARACTERISTICS	SOFTWARE QUALITY FACTORS
HUMAN LIVES AFFECTED	INTEGRITY RELIABILITY CORRECTNESS VERIFIABILITY, TESTABILITY SURVIVABILITY
LONG LIFE CYCLE	MAINTAINABILITY EXPANDABILITY, FLEXIBILITY
EXPERIMENTAL SYSTEM OR HIGH RATE OF CHANGE	EXPANDABILITY, FLEXIBILITY
EXPERIMENTAL TECHNOLOGY IN HARDWARE DESIGN	TRANSPORTABILITY
MANY CHANGES OVER LIFE CYCLE	REUSABILITY EXPANDABILITY, FLEXIBILITY
REAL TIME APPLICATION	EFFICIENCY RELIABILITY CORRECTNESS
ON-BOARD COMPUTER APPLICATION	EFFICIENCY RELIABILITY CORRECTNESS SURVIVABILITY
PROCESSING OF CLASSIFIED INFORMATION	INTEGRITY
INTERRELATED SYSTEMS	INTEROPERABILITY
INTERACTIVE SYSTEMS	USABILITY
BATCH SYSTEMS	EFFICIENCY RELIABILITY CORRECTNESS

TABLE 6-4
COMPLEMENTARY SOFTWARE QUALITY FACTORS

G-04768

QUALITY FACTOR SPECIFIED \ COMPLEMENTARY QUALITY FACTOR	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	INTEROPERABILITY	REUSABILITY	TRANSPORTABILITY
EFFICIENCY												
INTEGRITY			*			*		*				
RELIABILITY						*		*				
SURVIVABILITY			*			*		*				
USABILITY			*			*	*	*				
CORRECTNESS												
MAINTAINABILITY			*			*						
VERIFIABILITY, TESTABILITY			*			*	*					
EXPANDABILITY, FLEXIBILITY			*			*	*	*				
INTEROPERABILITY			*			*	*	*				
REUSABILITY			*			*	*	*				
TRANSPORTABILITY			*			*	*	*				

* = DEPENDENCY

Quality levels for the other specified factors are difficult to measure accurately when there are low quality levels for any of the four complementary factors. For example, if a high quality level for reliability is specified, high quality levels for correctness and verifiability should also be specified. This is because if the scores for reliability are high, but the software is incorrect or difficult to verify, the true reliability may be low due to incorrect software or uncertainty in verification. The complementary quality factors show that the attributes are not independent. Any project, regardless of the type of system or application, should consider the complementary factors in the quality specifications.

The criteria that are attributes of more than one quality factor are shared criteria. For example, Table 6-2 shows simplicity is a criterion for five of the factors. The beneficial effect of this is that these attributes are built into the software only once; likewise, assessment of these attributes need only be done once. Therefore, costs associated with specifying factors that share common criteria are generally less than costs associated with specifying factors that do not share criteria.

Assigning more than one quality factor to an APSE component can have either a beneficial or an adverse effect, depending on the combination of factors that have been specified. Some factors have criteria that conflict with another factor; and some have criteria that cooperate with another factor. Attribute criteria affecting other factors are shown in Table 6-5. Attributes that are basic criteria of a factor are identified with an "x"; criteria that are in a positive or beneficial relationship with another factor are identified with a "+"; and criteria that are in a negative or adverse relationship with another factor are identified with a "-". For example, operability is a criterion of usability and is shown to have a beneficial relationship with maintainability. The assertion is that the operability of usable software aids in software maintenance, even though it is not an essential characteristic of maintainability. The implication is that the effort to maintain software will be less if usability is also a specified quality. An example of a criterion with an adverse relationship is anomaly management. Anomaly management is a criterion of reliability and is shown to have a conflicting relationship with efficiency. The assertion is that the additional code required to perform anomaly management increases the runtime and requires additional memory, thus decreasing the potential efficiency. This implies that efficient use of resources will be more

TABLE 6-5

BENEFICIAL AND ADVERSE EFFECTS OF CRITERIA ON SOFTWARE QUALITY FACTORS

G-04767

ACQUISITION CONCERN	CRITERION/SECTION		PERFORMANCE 6.1: 1 EFFICIENCY	2 INTEGRITY	3 RELIABILITY	4 SURVIVABILITY	5 USABILITY	DESIGN 6.2: 1 CORRECTNESS	2 MAINTAINABILITY	3 VERIFIABILITY	ADAPTATION 6.3: 1 EXPANDABILITY	2 INTEROPERABILITY	3 REUSABILITY	4 TRANSPORTABILITY
PERFORMANCE	ACCURACY	6.4.1	-		X									
	ANOMALY MANAGEMENT (FAULT OR ERROR TOLERANCE, ROBUSTNESS	6.4.2	-		X	X	+							
	AUTONOMY	6.4.5				X								
	CAPACITY	6.4.6					X				+			
	COMMUNICATION EFFECTIVENESS	6.4.8	X						-	-				-
	COST	6.4.11					X							
	DISTRIBUTEDNESS	6.4.12		-		X					+			
	MATURITY	6.4.18					X							
	OPERABILITY (COMMUNICATIVENESS)	6.4.20	-				X		+	+				
	POWER	6.4.21					X							
	PROCESSING (EXECUTION) EFFECTIVENESS	6.4.22	X						-	-				-
	RECONFIGURABILITY	6.4.24				X			+		-		-	-
	REQUIRED CONFIGURATION	6.4.26					X							
	STORAGE EFFECTIVENESS	6.4.31	X						-	-				-
	SYSTEM ACCESSIBILITY	6.4.32	-	X										
	TRAINING	6.4.36					X							
DESIGN	COMPLETENESS	6.4.9						X	+				+	
	CONSISTENCY	6.4.10						X	X	+	+		+	
	TRACEABILITY	6.4.35						X	+	+	+		+	
	VISIBILITY (TEST AVAILABILITY)	6.4.38							X	X				
ADAPTATION	APPLICATION INDEPENDENCE	6.4.3											X	+
	AUGMENTABILITY	6.4.4									X			
	COMMONALITY (DATA AND COMMUNICATION)	6.4.7	-									X		
	DOCUMENT ACCESSIBILITY	6.4.13		-					+				X	
	FUNCTIONAL OVERLAP	6.4.14										X		
	FUNCTIONAL SCOPE	6.4.15											X	
	GENERALITY	6.4.16	-	-	-	-					X	-	X	
	REHOSTABILITY (INDEPENDENCE)	6.4.25	-									X	X	X
	RETARGETABILITY (INDEPENDENCE)	6.4.27	-								X	X	X	X
	SYSTEM CLARITY	6.4.33											X	
	SYSTEM COMPATIBILITY	6.4.34		-								X		
	VIRTUALITY	6.4.37	-								X			
GENERAL	GRANULARITY	6.4.17				X	X		X	X	X		X	
	MODULARITY	6.4.19	-			X			X	X	X	X	X	X
	PROPRIETARY RIGHTS	6.4.23							X		X			
	SELF-DESCRIPTIVENESS	6.4.28	-						X	X	X		X	X
	SIMPLICITY	6.4.29	-		X				X	X	X		X	
	SOFTWARE PRODUCTION VEHICLE(S)	6.4.30							X		X			

LEGEND

X = BASIC RELATIONSHIP

+ = POSITIVE EFFECT

- = NEGATIVE EFFECT

BLANK = NONE OR APPLICATION DEPENDENT

difficult to achieve if reliability is also a specified quality factor. Possible solutions to this conflict include:

- Budget and schedule to try to achieve high goals for both factors
- Lowering goals for one or the other factor, including decreasing emphasis on the specific criterion that conflicts and increasing emphasis on those that do not conflict
- Allocating more resources to the host system (e.g., more efficient processor or more memory) and possibly decreasing emphasis on high software quality goals.

Figure 6-1 shows how the attributes are related to other elements of the E&V Reference Manual and to elements in the E&V Guidebook. The various attributes that are useful in defining assessment objectives for APSEs or APSE components are described in this chapter.

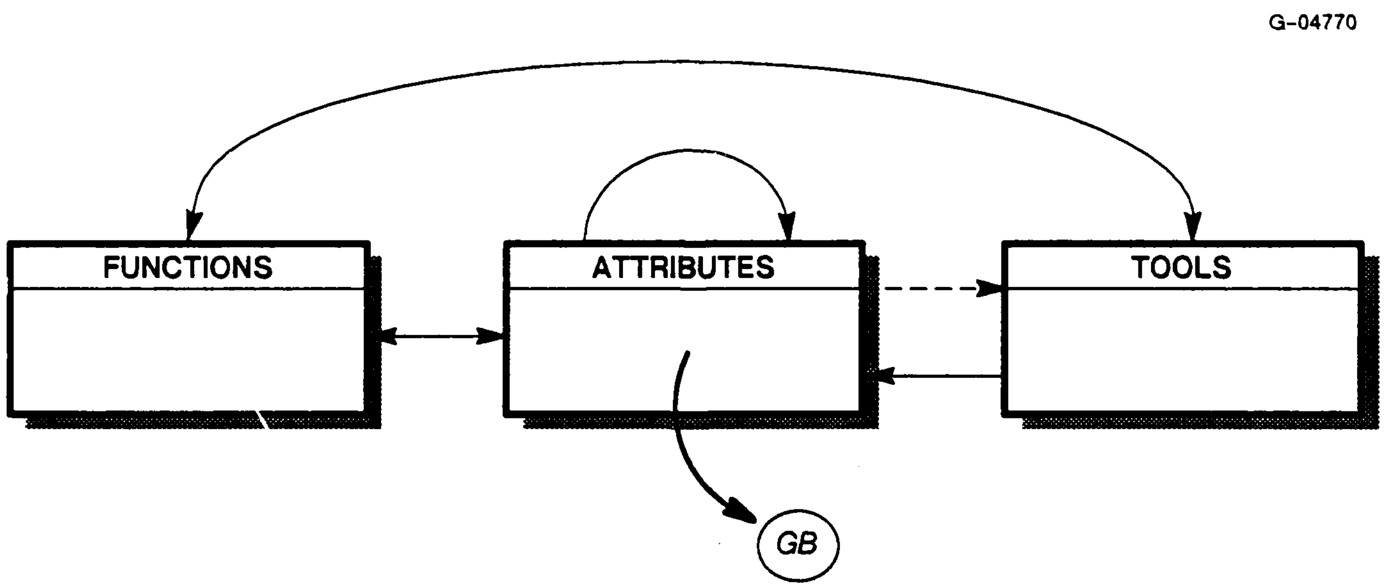


Figure 6-1 Attribute Relationships

6.1 PERFORMANCE

Description:

Performance quality factors deal both with the ability of the software to function and with error occurrences that affect software functioning. Low quality factors predict poor software performance. [@RADC 1985]

Cross References:

Software Quality Factors:

[Efficiency	6.1.1,
Integrity	6.1.2,
Reliability	6.1.3,
Survivability	6.1.4,
Usability	6.1.5]

6.1.1 Efficiency

Description:

Efficiency deals with utilization of resources. [@RADC 1985] The extent to which a component fulfills its purpose using a minimum of computing resources. Of course, many of the ways of coding efficiently are not necessarily efficient in the sense of being cost-effective, since transportability, maintainability, etc., may be degraded as a result. [@DACS 1979]

Cross References:

Acquisition Concern:	
[Performance	6.1]
Software-Oriented Criteria:	
[Communication Effectiveness	6.4.8,
Processing Effectiveness	6.4.22,
Storage Effectiveness	6.4.31]
Application/Environment Characteristics:	
[Real Time Application,	
On-Board Computer Application,	
Batch Systems]	
Complementary Software Quality Factors:	
Cooperating Criteria:	
Conflicting Criteria:	
[Accuracy	6.4.1,
Anomaly Management	6.4.2,
Commonality	6.4.7,
Generality	6.4.16,
Modularity	6.4.19,
Operability	6.4.20,
Reconfigurability	6.4.24,
Rehostability	6.4.25,
Retargetability	6.4.27,
Self-Descriptiveness	6.4.28,
Simplicity	6.4.29,
System Accessibility	6.4.32
Virtuality	6.4.37]

6.1.2 Integrity

Description:

Integrity deals with software failures due to unauthorized access. [@RADC 1985] The extent to which access to a component or associated data by unauthorized persons can be controlled. [@E&V Requirements]

Cross References:

Acquisition Concern:	
[Performance	6.1]
Software-Oriented Criteria:	
[System Accessibility	6.4.32]
Application/Environment Characteristics:	
[Human Lives Affected,	
Processing of Classified Information]	
Complementary Software Quality Factors:	
[Reliability	6.1.3,
Correctness	6.2.1,
Verifiability, Testability	6.2.3]
Cooperating Criteria:	
Conflicting Criteria:	
[Distributedness	6.4.12,
Document Accessibility	6.4.13,
Generality	6.4.16,
System Compatibility	6.4.34]

6.1.3 Reliability

Description:

Reliability concerns any software failure. [@RADC 1985] The extent to which a component can be expected to perform its intended functions in a satisfactory manner. [@DACS 1979]

Cross References:

Acquisition Concern:	
[Performance	6.1]
Software-Oriented Criteria:	
[Accuracy	6.4.1,
Anomaly Management	6.4.2,
Simplicity	6.4.29]
Application/Environment Characteristics:	
[Human Lives Affected,	
Real Time Application,	
On-Board Computer Application,	
Batch Systems]	
Complementary Software Quality Factors:	
[Correctness	6.2.1,
Verifiability, Testability	6.2.3]
Cooperating Criteria:	
Conflicting Criteria:	
[Generality	6.4.16]

6.1.4 Survivability

Description:

Survivability deals with the continued performance of software (e.g., in a degraded *mode) even when a portion of the system has failed.* [@RADC 1985]

Cross References:

Acquisition Concern:	
[Performance	6.1]
Software-Oriented Criteria:	
[Anomaly Management	6.4.2,
Autonomy	6.4.5,
Distributedness	6.4.12,
Granularity	6.4.17
Modularity	6.4.19,
Reconfigurability	6.4.24]
Application/Environment Characteristics:	
[Human Lives Affected,	
On-Board Computer Application]	
Complementary Software Quality Factors:	
[Reliability	6.1.3,
Correctness	6.2.1,
Verifiability, Testability	6.2.3]
Cooperating Criteria:	
Conflicting Criteria:	
[Generality	6.4.16]

6.1.5 Usability

Description:

Extent to which resources required to acquire, install, learn, operate, prepare input for, and interpret output of a component are minimized.

Cross References:

Acquisition Concern:	
[Performance	6.1]
Software-Oriented Criteria:	
[Capacity	6.4.6,
Cost	6.4.11,
Granularity	6.4.17,
Maturity	6.4.18,
Operability	6.4.20,
Power	6.4.21,
Required Configuration	6.4.26,
Training	6.4.36]
Application/Environment Characteristics:	
[Interactive Systems]	
Complementary Software Quality Factors:	
[Reliability	6.1.3,
Correctness	6.2.1,
Maintainability	6.2.2,
Verifiability, Testability	6.2.3]
Cooperating Criteria:	
[Anomaly Management	6.4.2]
Conflicting Criteria:	

6.2 DESIGN

Description:

Design quality factors deal mainly with software failure and correction. Low quality levels usually result in repeating a portion of the development process (e.g., redesign, recode, reverify); hence the term design. [@RADC 1985]

Cross References:

Software Quality Factors:

[Correctness	6.2.1,
Maintainability	6.2.2,
Verifiability, Testability	6.2.3]

6.2.1 Correctness

Description:

The extent to which software design and implementation conform to specifications and standards. Criteria of correctness deal exclusively with design and documentation formats. [@RADC 1985] Agreement between a component's total response and the stated response in the functional specification (functional correctness), and/or between the component as coded and the programming specification (algorithmic correctness). [@DACS 1979]

Cross References:

Acquisition Concern:

[Design	6.2]

Software-Oriented Criteria:

[Completeness	6.4.9,
Consistency	6.4.10,
Traceability	6.4.35]

Application/Environment Characteristics:
[Human Lives Affected,
Real Time Application,
On-Board Computer Application,
Batch Systems]

Complementary Software Quality Factors:

Cooperating Criteria:

Conflicting Criteria:

6.2.2 Maintainability

Description:

The ease of effort in locating and fixing software failures. [@RADC 1985] The extent to which a component facilitates updating to satisfy new requirements or to correct deficiencies. This implies that the component is understandable, testable, and modifiable. [@DACS 1979]

Cross References:

Acquisition Concern:	
[Design	6.2]
Software-Oriented Criteria:	
[Consistency	6.4.10,
Granularity	6.4.17
Modularity	6.4.19,
Proprietary Rights	6.4.23
Self-Descriptiveness	6.4.28,
Simplicity	6.4.29,
Software Production Vehicle(s)	6.4.30,
Visibility	6.4.38]
Application/Environment Characteristics:	
[Long Life Cycle]	
Complementary Software Quality Factors:	
[Reliability	6.1.3,
Correctness	6.2.1]
Cooperating Criteria:	
[Completeness	6.4.9,
Document Accessibility	6.4.13,
Operability	6.4.20,
Reconfigurability	6.4.24,
Traceability	6.4.35]
Conflicting Criteria:	
[Communication Effectiveness	6.4.8,
Processing Effectiveness	6.4.22,
Storage Effectiveness	6.4.31]

6.2.3 Verifiability, Testability

Description:

Software design characteristics affecting the effort to verify software operation and performance. [@RADC 1985] The extent to which a component facilitates the establishment of verification criteria and supports evaluation of its performance. This implies that requirements are matched to specific modules, or diagnostic capabilities are provided, etc. [@DACS 1979]

Cross References:

Acquisition Concern:	
[Design	6.2]
Software-Oriented Criteria:	
[Granularity	6.4.17,
Modularity	6.4.19,
Self-Descriptiveness	6.4.28,
Simplicity	6.4.29,
Visibility	6.4.38]
Application/Environment Characteristics:	
[Human Lives Affected]	
Complementary Software Quality Factors:	
[Reliability	6.1.3,
Correctness	6.2.1,
Maintainability	6.2.2]
Cooperating Criteria:	
[Consistency	6.4.10,
Operability	6.4.20,
Traceability	6.4.35]
Conflicting Criteria:	
[Communication Effectiveness	6.4.8,
Processing Effectiveness	6.4.22,
Storage Effectiveness	6.4.31]

6.3 ADAPTATION

Description:

Adaptation quality factors deal mainly with using software beyond its original requirements, such as extending or expanding capabilities and adapting for use in another application or environment. Low quality levels predict relatively high costs for new software use. [@RADC 1985]

Cross References:

Software Quality Factors:

[Expandability, Flexibility	6.3.1,
Interoperability	6.3.2,
Reusability	6.3.3,
Transportability	6.3.4]

6.3.1 Expandability, Flexibility

Description:

The effort in increasing software capabilities or performance or to accommodate changes in requirements. [@RADC 1985] The extent to which a component allows new capabilities to be added and existing capabilities to be easily tailored to user needs. [@DACS 1979]

Cross References:

Acquisition Concern:
[Adaptation 6.3]

Software-Oriented Criteria:

[Augmentability	6.4.4,
Generality	6.4.16,
Granularity	6.4.17,
Modularity	6.4.19,
Proprietary Rights	6.4.23,
Retargetability	6.4.27,
Self-Descriptiveness	6.4.28,
Simplicity	6.4.29,
Software Production Vehicle(s)	6.4.30,
Virtuality	6.4.37]

Application/Environment Characteristics:
[Long Life Cycle,
Experimental System or High Rate of Change,
Many Changes over Life Cycle]

Complementary Software Quality Factors:

[Reliability	6.1.3,
Correctness	6.2.1,
Maintainability	6.2.2,
Verifiability, Testability	6.2.3]

Cooperating Criteria:

[Capacity	6.4.6,
Consistency	6.4.10,
Distributedness	6.4.12,
Traceability	6.4.35]

Conflicting Criteria:
[Reconfigurability 6.4.24]

6.3.2 Interoperability

Description:

The effort in coupling software of one system to software of one or more other systems. [@RADC 1985] The ability of APSEs to exchange data base objects and their relationships in forms usable by components and user programs without conversion. Interoperability is measured by the degree to which this exchange can be accomplished without conversion. [@E&V Requirements]

Cross References:

Acquisition Concern:

[Adaptation	6.3]

Software-Oriented Criteria:

[Commonality	6.4.7,
Functional Overlap	6.4.14,
Modularity	6.4.19,
Rehostability	6.4.25,
Retargetability	6.4.27,
System Compatibility	6.4.34]

Application/Environment Characteristics:
[Interrelated Systems]

Complementary Software Quality Factors:

[Reliability	6.1.3,
Correctness	6.2.1,
Maintainability	6.2.2,
Verifiability, Testability	6.2.3]

Cooperating Criteria:

[Generality	6.4.16]

Conflicting Criteria:

6.3.3 Reusability

Description:

The extent to which a component can be adapted for use in another application. [@RADC 1985]

Cross References:

Acquisition Concern:	
[Adaptation	6.3]
Software-Oriented Criteria:	
[Application Independence	6.4.3,
Document Accessibility	6.4.13,
Functional Scope	6.4.15,
Generality	6.4.16,
Granularity	6.4.17
Modularity	6.4.19,
Rehostability	6.4.25,
Retargetability	6.4.27,
Self-Descriptiveness	6.4.28,
Simplicity	6.4.29,
System Clarity	6.4.33]
Application/Environment Characteristics:	
[Many Changes over Life Cycle]	
Complementary Software Quality Factors:	
[Reliability	6.1.3,
Correctness	6.2.1,
Maintainability	6.2.2,
Verifiability, Testability	6.2.3]
Cooperating Criteria:	
[Completeness	6.4.9,
Consistency	6.4.10,
Traceability	6.4.35]
Conflicting Criteria:	
[Reconfigurability	6.4.24]

6.3.4 Transportability

Description:

The extent to which a component can be adapted for use in another environment (e.g., different host or target hardware, operating system, APSE). [@RADC 1985] The ability of a component to be installed on a different APSE without change in functionality. Transportability is measured in the degree to which this installation can be accomplished without reprogramming. [@E&V Requirements]

Cross References:

Acquisition Concern:
[Adaptation 6.3]

Software-Oriented Criteria:
[Modularity 6.4.19,
Rehostability 6.4.25,
Retargetability 6.4.27,
Self-Descriptiveness 6.4.28]

Application/Environment Characteristics:
[Experimental Technology in Hardware Design]

Complementary Software Quality Factors:
[Reliability 6.1.3,
Correctness 6.2.1,
Maintainability 6.2.2,
Verifiability, Testability 6.2.3]

Cooperating Criteria:
[Application Independence 6.4.3]

Conflicting Criteria:
[Communication Effectiveness 6.4.8,
Processing Effectiveness 6.4.22,
Reconfigurability 6.4.24,
Storage Effectiveness 6.4.31]

6.4 SOFTWARE-ORIENTED CRITERIA

6.4.1 Accuracy

Description:

Those characteristics of software which provide the required precision in calculations and outputs. [@RADC 1985]

Cross References:

Software Quality Factors:
[Reliability 6.1.3]

Beneficial Quality Factors:

Adverse Quality Factors:
[Efficiency 6.1.1]

Guidebook References:

6.4.2 Anomaly Management, Fault or Error Tolerance, Robustness

Description:

Those characteristics of software which provide for continuity of operations under and recovery from non-nominal conditions. [@RADC 1985] The protection of a component from itself, user errors, and system errors. The ability to recover and provide meaningful diagnostics in the event of unforeseen situations. A robust routine will avoid failing for input values where the desired output is well-defined, but the intermediate results of a straightforward implementation may cause the routine to fail. [@E&V Requirements]

Cross References:

Software Quality Factors:	
[Reliability	6.1.3,
Survivability	6.1.4]
Beneficial Quality Factors:	
[Usability	6.1.5]
Adverse Quality Factors:	
[Efficiency	6.1.1]

Guidebook References:

6.4.3 Application Independence

Description:

Those characteristics of software which determine its non-dependency on database system, microcode, computer architecture, and algorithms. [@RADC 1985]

Cross References:

Software Quality Factors:
[Reusability 6.3.3]

Beneficial Quality Factors:
[Transportability 6.3.4]

Adverse Quality Factors:

Guidebook References:

6.4.4 Augmentability

Description:

Those characteristics of software which provide for expansion of capability for functions and data. [@RADC 1985]

Cross References:

Software Quality Factors:
[Expandability, Flexibility 6.3.1]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.5 Autonomy

Description:

Those characteristics of software which determine its non-dependency on interfaces and functions. [@RADC 1985]

Cross References:

Software Quality Factors:
[Survivability 6.1.4]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.6 Capacity

Description:

The upper and lower limits of the functions implemented by a tool. [@E&V Requirements]

Cross References:

Software Quality Factors:
[Usability 6.1.4]

Beneficial Quality Factors:
[Expandability, Flexibility 6.3.1]

Adverse Quality Factors:

Guidebook References:

[Compilation 7.1.6.7, @GB: IDA Benchmarks 6.1]

6.4.7 Commonality (Data and Communication)

Description:

Those characteristics of software which provide for the use of interface standards for protocols, routines, and data representations. [@RADC 1985] The set of assumptions made by the component and made about the component by the remaining components and the system in which it appears. Software components have control, data, and service interfaces. Included in this attribute is conformance to any existing pertinent interface standards such as the CAIS. [@DACS 1979]

Cross References:

Software Quality Factors:
[Interoperability 6.3.2]

Beneficial Quality Factors:

Adverse Quality Factors:
[Efficiency 6.1.1]

Guidebook References:

6.4.8 Communication Effectiveness

Description:

Those characteristics of the software which provide for minimum utilization of communications resources in performing functions. [@RADC 1985]

Cross References:

Software Quality Factors:	
[Efficiency	6.1.1]
Beneficial Quality Factors:	
Adverse Quality Factors:	
[Maintainability	6.2.2,
Verifiability, Testability	6.2.3,
Transportability	6.3.4]

Guidebook References:

6.4.9 Completeness

Description:

The extent to which a component provides the complete set of operations necessary to perform a function. [@E&V Requirements]

Cross References:

Software Quality Factors:

[Correctness	6.2.1]

Beneficial Quality Factors:

[Maintainability	6.2.2,
Reusability	6.3.3]

Adverse Quality Factors:

Guidebook References:

[Compilation	7.1.6.7, @GB: ACVC	8.1;
Kernel	7.2.3.3, @GB: CIVC	8.2]

6.4.10 Consistency

Description:

Those characteristics of software which provide for uniform design and implementation techniques and notation. [@RADC 1985]

Cross References:

Software Quality Factors:	
[Correctness	6.2.1,
Maintainability	6.2.2]
Beneficial Quality Factors:	
[Verifiability, Testability	6.2.3,
Expandability, Flexibility	6.3.1,
Reusability	6.3.3]
Adverse Quality Factors:	

Guidebook References:

6.4.11 Cost

Description:

The cost of a complete component or the costs of features of a component. The cost of a component may vary depending on delivery with source code or object code only (for example). Other cost considerations are installation, user assistance, and maintenance support. [@E&V Requirements]

Cross References:

Software Quality Factors:
[Usability 6.1.5]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.12 Distributedness

Description:

Those characteristics of software which determine the degree to which software functions are geographically or logically separated within the system. [@RADC 1985]

Cross References:

Software Quality Factors:	
[Survivability	6.1.4]
Beneficial Quality Factors:	
[Expandibility, Flexibility	6.3.1]
Adverse Quality Factors:	
[Integrity	6.1.2]

Guidebook References:

6.4.13 Document Accessibility

Description:

Those characteristics of software which provide for easy access to software and selective use of its components. [@RADC 1985]

Cross References:

Software Quality Factors:
[Reusability 6.3.3]

Beneficial Quality Factors:
[Maintainability 6.2.2]

Adverse Quality Factors:
[Integrity 6.1.2]

Guidebook References:

6.4.14 Functional Overlap

Description:

Those characteristics of software which provide common functions to both systems. [@RADC 1985]

Cross References:

Software Quality Factors:
[Interoperability 6.3.2]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.15 Functional Scope

Description:

Those characteristics of software which provide commonality of functions among applications. [@RADC 1985]

Cross References:

Software Quality Factors:
[Reusability 6.3.3]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.16 Generality

Description:

Those characteristics of software which provide breadth to the functions performed with respect to the application. [@RADC 1985]

Cross References:

Software Quality Factors:	
[Expandability, Flexibility	6.3.1,
Reusability	6.3.3]
Beneficial Quality Factors:	
[Interoperability	6.3.2]
Adverse Quality Factors:	
[Efficiency	6.1.1,
Integrity	6.1.2,
Reliability	6.1.3,
Survivability	6.1.4]

Guidebook References:

6.4.17 Granularity

Description:

The degree to which a component has separate limited functions that are composable, user selectable, and communicate through a common data base. [@E&V Requirements]

Cross References:

Software Quality Factors:

[Survivability	6.1.4,
Usability	6.1.5,
Maintainability	6.2.2,
Verifiability, Testability	6.2.3,
Expandability, Flexibility	6.3.1,
Reusability	6.3.3]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.18 Maturity

Description:

The extent to which a component has been used in the development of deliverable software by typical users and to which the feedback from that use has been reflected in modifications to the component. [@E&V Requirements]

Cross References:

Software Quality Factors:
[Usability 6.1.5]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.19 Modularity

Description:

Those characteristics of software which provide a structure of highly cohesive components with optimum coupling. [@RADC 1985] The extent to which a component is implemented in a hierarchical structure in which identifiable functions are isolated in separate compilation units.

Cross References:

Software Quality Factors:

[Survivability	6.1.4,
Maintainability	6.2.2,
Verifiability, Testability	6.2.3,
Expandability, Flexibility	6.3.1,
Interoperability	6.3.2,
Reusability	6.3.3,
Transportability	6.3.4]

Beneficial Quality Factors:

Adverse Quality Factors:

[Efficiency	6.1.1]

Guidebook References:

6.4.20 Operability, Communicativeness

Description:

Those characteristics of software which determine operations and procedures concerned with operation of software and which provide useful inputs and outputs which can be assimilated. [@RADC 1985] The user/component dialog established to control the execution of the component. This is driven by the set of assumptions made by the component and made about the component by the persons who use it.

Cross References:

Software Quality Factors:	
[Usability	6.1.5]
Beneficial Quality Factors:	
[Maintainability	6.2.2,
Verifiability, Testability	6.2.3]
Adverse Quality Factors:	
[Efficiency	6.1.1]

Guidebook References:

[@GB: Ada-Europe User Interface Checklist	5.2.3;
@GB: Weiderman's Human Interface Checklist	5.3.1]

6.4.21 Power

Description:

The extent to which a component has capabilities, such as default options and wild card operations, that contribute to the effectiveness of the user. [@E&V Requirements]

Cross References:

Software Quality Factors:
[Usability 6.1.5]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

[Text Editing	7.1.1.1,	
@GB: Text Editing Checklist		5.1.1;
Assembling	7.1.6.6,	
@GB: Assembling Checklist		5.1.2;
Compilation	7.1.6.7,	
@GB: Compilation Checklist		5.1.3;
Linking/Loading	7.1.6.13,	
@GB: Linking/Loading Checklist		5.1.4;
Data Base Management	7.2.1.1,	
@GB: Data Base Management Checklist		5.1.5;
File Management	7.2.1.3,	
@GB: File Management Checklist		5.1.6;
Electronic Mail	7.2.1.4,	
@GB: Electronic Mail Checklist		5.1.7;
Program Library Management	7.2.1.7,	
@GB: Program Library Management Checklist		5.1.8;
Performance Monitoring	7.2.1.10,	

@GB: Performance Monitoring Checklist		5.1.9;
Scheduling	7.2.2.3,	
@GB: Scheduling Checklist		5.1.10;
Tracking	7.2.2.6,	
@GB: Tracking Checklist		5.1.11;
Configuration Management	7.2.2.7,	
@GB: Configuration Management Checklist		5.1.12;
Input/Output Support	7.2.3.2,	
@GB: Input/Output Support Checklist		5.1.13;
Import/Export	7.2.3.6,	
@GB: Import/Export Checklist		5.1.14;
Requirements Prototyping	7.3.2.2,	
@GB: Requirements Prototyping Checklist		5.1.15;
Simulation and Modelling	7.3.2.3,	
@GB: Simulation and Modelling Checklist		5.1.16;
Debugging	7.3.2.5,	
@GB: Debugging Checklist		5.1.17;
Executable Assertion Checking	7.3.2.6,	
@GB: Executable Assertion Checking Checklist		5.1.18;
Mutation Analysis	7.3.2.9,	
@GB: Mutation Analysis Checklist		5.1.19;
Testing	7.3.2.10,	
@GB: Testing Checklist		5.1.20;
Emulation	7.3.2.13,	
@GB: Emulation Checklist		5.1.21;
Timing	7.3.2.14,	
@GB: Timing Checklist		5.1.22;
Tuning	7.3.2.15,	
@GB: Tuning Checklist		5.1.23;
Real Time Analysis	7.3.2.17,	
@GB: Real Time Analysis Checklist		5.1.24]

6.4.22 Processing (Execution) Effectiveness

Description:

Those characteristics of the software which provide for minimum utilization of processing resources in performing functions. [@RADC 1985] The choice between alternative algorithms based on those taking the least amount of time. [@DACS 1979]

Cross References:

Software Quality Factors:
[Efficiency 6.1.1]

Beneficial Quality Factors:

Adverse Quality Factors:
[Maintainability 6.2.2,
Verifiability, Testability 6.2.3,
Transportability 6.3.4]

Guidebook References:

*[Compilation	7.1.6.7, @GB: IDA Benchmarks	6.1;
* Compilation	7.1.6.7, @GB: ACEC	6.2]

*NOTE: Normally, the concern of evaluation is the quality of a tool. However, this evaluation technique focuses on the <u>products</u> of the tool rather than the tool itself.

6.4.23 Proprietary Rights

Description:

Restrictions on the release, distribution, or use of a component. This includes so called "data rights" restrictions. [@E&V Requirements]

Cross References:

Software Quality Factors:

[Maintainability	6.2.2,
Expandability, Flexibility	6.3.1]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.24 Reconfigurability

Description:

Those characteristics of software which provide for continuity of system operation when one or more processors, storage units, or communication links fails. [@RADC 1985]

Cross References:

Software Quality Factors:

[Survivability	6.1.4]

Beneficial Quality Factors:

[Maintainability	6.2.2]

Adverse Quality Factors:

[Efficiency	6.1.1,
Expandability, Flexibility	6.3.1,
Reusability	6.3.3,
Transportability	6.3.4]

Guidebook References:

6.4.25 Rehostability

Description:

The ability of an APSE component to be installed on a different host or different operating system with needed reprogramming localized to the KAPSE or machine dependencies. [@E&V Requirements]

Cross References:

Software Quality Factors:

[Interoperability	6.3.2,
Reusability	6.3.3,
Transportability	6.3.4]

Beneficial Quality Factors:

Adverse Quality Factors:

[Efficiency	6.1.1]

Guidebook References:

[@GB: Ada-Europe Host and Target Checklist	5.2.1,
@GB: Ada-Europe Machine-Specific Characteristics Checklist	5.2.2]

6.4.26 Required Configuration

Description:

The amount and types of hardware or software facilities needed for the operation of a component. This includes primary and secondary storage and any other required resources. [@E&V Requirements]

Cross References:

Software Quality Factors:
[Usability 6.1.5]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.27 Retargetability

Description:

The ability of an APSE component to accomplish its function with respect to another target. The component may require modification. [@E&V Requirements]

Cross References:

Software Quality Factors:

[Expandability, Flexibility	6.3.1,
Interoperability	6.3.2,
Reusability	6.3.3,
Transportability	6.3.4]

Beneficial Quality Factors:

Adverse Quality Factors:

[Efficiency	6.1.1]

Guidebook References:

[@GB: Ada-Europe Host and Target Checklist	5.2.1,
@GB: Ada-Europe Machine-Specific Characteristics Checklist	5.2.2]

6.4.28 Self-Descriptiveness

Description:

Those characteristics of software which provide explanation of the implementation of functions. [@RADC 1985] The technical data, including on-line, documentation, listings, and printouts, which serve the purpose of elaborating the design or details of a component. [@DACS 1979]

Cross References:

Software Quality Factors:

[Maintainability	6.2.2,
Verifiability, Testability	6.2.3,
Expandability, Flexibility	6.3.1,
Reusability	6.3.3,
Transportability	6.3.4]

Beneficial Quality Factors:

Adverse Quality Factors:

[Efficiency	6.1.1]

Guidebook References:

6.4.29 Simplicity

Description:

Those characteristics of software which provide for definition and implementation of functions in the most non-complex and understandable manner. [@RADC 1985] A simple program style is one which makes the program, as a whole, easy to understand. Other things being equal, the style is concise and straightforward. It makes use of process, procedural, and data abstraction, as appropriate, to present a clear exposition of the intent.

Cross References:

Software Quality Factors:	
[Reliability	6.1.3,
Maintainability	6.2.2,
Verifiability, Testability	6.2.3,
Expandability, Flexibility	6.3.1,
Reusability	6.3.3]

Beneficial Quality Factors:

Adverse Quality Factors:	
[Efficiency	6.1.1]

Guidebook References:

6.4.30 Software Production Vehicle(s)

Description:

The methodology(ies), language(s), and technique(s) used to produce the software related to a component. [@E&V Requirements]

Cross References:

Software Quality Factors:

[Maintainability	6.2.2,
Expandability, Flexibility	6.3.1]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.31 Storage Effectiveness

Description:

Those characteristics of the software which provide for minimum utilization of storage resources in performing functions. [@RADC 1985] The choice between alternative source code constructions based on those taking the minimum number of words of object code or in which the information-packing . . . is high. [@DACS 1979]

Cross References:

Software Quality Factors:	
[Efficiency	6.1.1]
Beneficial Quality Factors:	
Adverse Quality Factors:	
[Maintainability	6.2.2,
Verifiability, Testability	6.2.3,
Transportability	6.3.4]

Guidebook References:

*[Compilation	7.1.6.7, @GB: IDA Benchmarks	6.1;
* Compilation	7.1.6.7, @GB: ACEC	6.2]

*NOTE: Normally, the concern of evaluation is the quality of a tool. However, this evaluation technique focuses on the products of the tool rather than the tool itself.

6.4.32 System Accessibility

Description:

Those characteristics of software which provide for control and audit of access to the software and data. [@RADC 1985]

Cross References:

Software Quality Factors:
[Integrity 6.1.2]

Beneficial Quality Factors:

Adverse Quality Factors:
[Efficiency 6.1.1]

Guidebook References:

6.4.33 System Clarity

Description:

Those characteristics of software which provide for clear description of program structure in a non-complex and understandable manner. [@RADC 1985]

Cross References:

Software Quality Factors:
[Reusability 6.3.3]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.34 System Compatibility

Description:

Those characteristics of software which provide the hardware, software, and communication compatibility of two systems. [@RADC 1985]

Cross References:

Software Quality Factors:
[Interoperability 6.3.2]

Beneficial Quality Factors:

Adverse Quality Factors:
[Integrity 6.1.2]

Guidebook References:

6.4.35 Traceability

Description:

Those characteristics of software which provide a thread of origin from the implementation to the requirements with respect to the specified development envelope and operational environment. [@RADC 1985]

Cross References:

Software Quality Factors:	
[Correctness	6.2.1]
Beneficial Quality Factors:	
[Maintainability	6.2.2,
Verifiability, Testability	6.2.3,
Expandability, Flexibility	6.3.1,
Reusability	6.3.3]
Adverse Quality Factors:	

Guidebook References:

6.4.36 Training

Description:

Those characteristics of software which provide transition from current operation and provide initial familiarization. [@RADC 1985] The extent to which training and other user help is available from the vendor of a component or from the component itself, including on-line, documentation, listings, and printouts, which serve the purpose of providing operating instructions for using the component to obtain desired results. [@DACS 1979]

Cross References:

Software Quality Factors:
[Usability 6.1.5]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

6.4.37 Virtuality

Description:

Those characteristics of software which present a system that does not require user knowledge of the physical, logical, or topological characteristics. [@RADC 1985]

Cross References:

Software Quality Factors:
[Expandability, Flexibility 6.3.1]

Beneficial Quality Factors:

Adverse Quality Factors:
[Efficiency 6.1.1]

Guidebook References:

6.4.38 Visibility, Test Availability

Description:

Those characteristics of software which provide status monitoring of the development and operation. [@RADC 1985] The availability of tests that verify the correctness or effectiveness of a component function or feature. These tests may also verify proper response for an incorrect input or technique. [@E&V Requirements]

Cross References:

Software Quality Factors:	
[Maintainability	6.2.2,
Verifiability, Testability	6.2.3]

Beneficial Quality Factors:

Adverse Quality Factors:

Guidebook References:

THIS PAGE LEFT INTENTIONALLY BLANK

7. FUNCTIONS

This chapter provides a functional taxonomy used for characterizing the functional capabilities of a tool, toolset, or APSE. At the highest level, functions are classified into the following three taxa, which are further decomposed as described below:

- Transformation
- Management
- Analysis.

This type of functional classification scheme was first described in the NBS Taxonomy report [@Houghton 1983] and later elaborated in the SEE Taxonomy [@Kean 1985]. The E&V Team has followed the general scheme suggested in the above-referenced documents, but with some further elaboration and modifications designed to make the taxonomy compatible with the rest of the multiple indexing scheme used in other parts of this manual.

Figure 7-1 shows how the functions are related to other elements of the E&V Reference Manual and to elements in the E&V Guidebook. The various functions that are useful in defining assessment objectives for APSEs or APSE components are described in this chapter. The relationships between tools and functions given in this chapter are typical (or traditional) relationships. The real capabilities should be determined by the tool specifications, marketing claims, or the like.

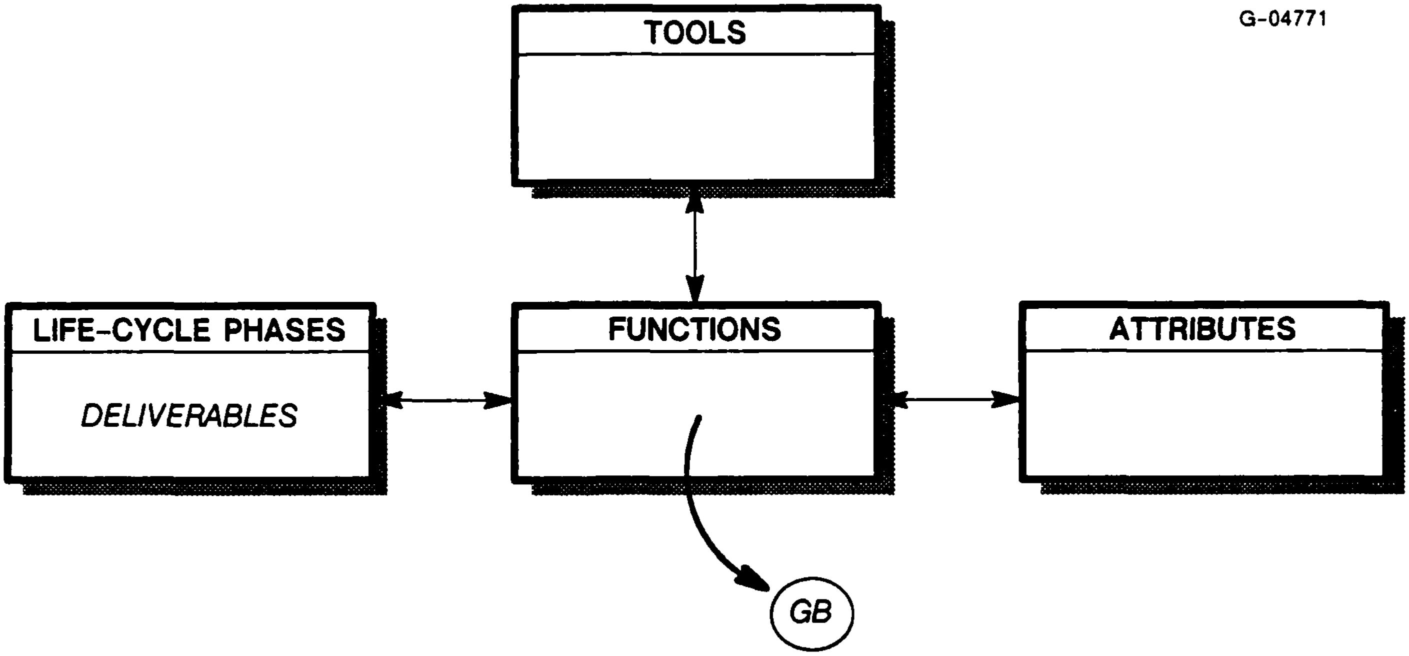
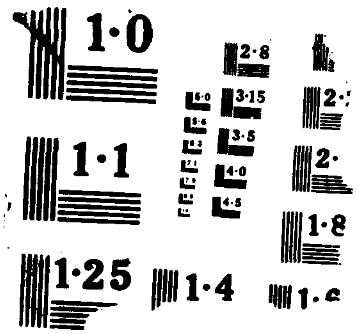


Figure 7-1 Function Relationships

AD-A193 309 E&V (EVALUATION AND VALIDATION) MANUAL VERSION 10(U) 3/4
ADA JOINT PROGRAM OFFICE ARLINGTON VA 29 DEC 87
TASC-TR-5234-3

UNCLASSIFIED F/G 12/5 NL


1·0
2·8
3·15
1·1
3·5
4·0
4·5
1·8
1·25
1·4

7.1 TRANSFORMATION

Description:

Transformation features describe how the subject is manipulated to accommodate the user's needs. They describe what transformations take place as input to the tool is processed. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.1.1 Editing

Description:

Selective revision of computer-resident data. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.1.1.1 Text

Description:

Editing capabilities provided for textual data. [@Kean 1985]

Cross References:

Life Cycle Phases:

[System Concepts	4.1.2,
System Requirements Analysis	4.2.2,
Software Requirements Analysis	4.3.2,
Preliminary Design	4.4.2,
Detailed Design	4.5.2,
Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2,
Global	4.12.2]

Tools:

[Syntax-Directed Editor	5.3.2,
Word Processor	5.4.2]

Guidebook References:

[Power 6.4.21,
@GB: Text Editing Checklist 5.1.1]

7.1.1.2 Data

Description:

Editing capabilities provided for data in internal (machine) format.

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.1.1.3 Graphics

Description:

Editing capabilities provided for graphical data. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.2]

Tools:

Guidebook References:

7.1.2 Formatting

Description:

Arranging data according to predefined and/or user-defined conventions. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

[Formatter	5.4.5,
Spreadsheet	5.5.1]

Guidebook References:

7.1.2.1 MIL-STD Format

Description:

The formatting of documents as specified by the MIL-STD(s).

Life Cycle Phases:
[Global 4.12.2]

Cross References:

Tools:

Guidebook References:

7.1.2.2 Table Of Contents

Description:

The production of a table of contents by examining the contents to be placed within the document.

Cross References:

Life Cycle Phases:
[Global 4.12.2]

Tools:

Guidebook References:

7.1.2.3 Predefined and User-Defined Forms

Description:

The ability to call up or define standard formats such as those that are mentioned in DoD-STD-2167.

Cross References:

Life Cycle Phases:

[System Requirements Analysis	4.2.2,
Software Requirements Analysis	4.3.1, 4.3.2,
Preliminary Design	4.4.1 - 4.4.4,
Detailed Design	4.5.1 - 4.5.4,
Coding And Unit Testing	4.6.1 - 4.6.4,
CSC Integration And Testing	4.7.1 - 4.7.4,
CSCI Testing	4.8.1 - 4.8.4,
Change Requirements	4.11.2]

Tools:

Guidebook References:

7.1.3 On-Line Assistance Processing

Description:

User interface that is part of the input/output process of a programming support environment (e.g., command assistance, assistance, on-line tutoring, definition assistance, etc.). [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.2]

Tools:
[On-Line Assistance Processor 5.1.8]

Guidebook References:

7.1.4 Sort/Merge

Description:

The process of arranging data in a specific order (e.g., alphabetical order). [@Kean 1985]

Cross References:

Life Cycle Phases:

[Global	4.12.2]

Tools:

[Address Book	5.5.3,
Phone Book	5.5.5,
Dictionary	5.5.8]

Guidebook References:

7.1.5 Graphics Generation

Description:

The input, construction, storage, retrieval, manipulation, alteration, and analysis of pictorial data (e.g., generation of system architectures, software designs, financial analysis, maps, graphs, etc.). [@Kean 1985]

Cross References:

Life Cycle Phases:

[Global	4.12.2]

Tools:

[Graphics Generator	5.4.4]

Guidebook References:

7.1.6 Translation

Description:

The conversion from one language form to another. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.1.6.1 Systems Requirements

Description:

The processing of formal systems requirements statements into an internal data base representation for subsequent use. [@Kean 1985]

Cross References:

Life Cycle Phases:
[System Requirements Analysis 4.2.2]

Tools:

Guidebook References:

7.1.6.2 Software Requirements

Description:

The processing of formal software requirements statements into an internal data base representation for subsequent use. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Software Requirements Analysis 4.3.2]

Tools:

Guidebook References:

7.1.6.3 Requirements To Natural Language

Description:

The transformation of formal requirements language constructs into English language text. [@Kean 1985]

Cross References:

Life Cycle Phases:

[System Requirements Analysis	4.2.2,
Software Requirements Analysis	4.3.2]

Tools:

Guidebook References:

7.1.6.4 Preliminary Design

Description:

The processing of formal preliminary design statements into an internal data base representation for subsequent use. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Preliminary Design 4.4.2]

Tools:

Guidebook References:

7.1.6.5 Detailed Design

Description:

The processing of formal detailed design statements into an internal data base representation for subsequent use. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Detailed Design 4.5.2]

Tools:

Guidebook References:

7.1.6.6 Assembling

Description:

Translating a program expressed in an assembly language into object code. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

[Assembler	5.3.4]

Guidebook References:

[Power	6.4.21, @GB: Assembling Checklist	5.1.2]

7.1.6.7 Compilation

Description:

Translating a computer program expressed in a procedural or problem-oriented language into object code. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

[Compiler	5.3.3]

Guidebook References:

[Capacity	6.4.6 , @GB: IDA Benchmarks	6.1;
Completeness	6.4.9, @GB: ACVC	8.1;
Power	6.4.21, @GB: Compilation Checklist	5.1.3;
*Processing Effectiveness	6.4.22, @GB: IDA Benchmarks	6.1;
*Processing Effectiveness	6.4.22, @GB: ACEC	6.2;
*Storage Effectiveness	6.4.31, @GB: IDA Benchmarks	6.1;
*Storage Effectiveness	6.4.31, @GB: ACEC	6.2]

*NOTE: Normally, the concern of evaluation is the quality of a tool. However, this evaluation technique focuses on the product of the tool rather than the tool itself.

7.1.6.8 Conversion

Description:

Modifying an existing program to enable it to operate with similar functional capabilities in a different environment. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

Guidebook References:

7.1.6.9 Macro Expansion

Description:

Augmenting instructions in a source language with user defined sequences of instructions in the same source language. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

Guidebook References:

7.1.6.10 Structure Preprocessing

Description:

Translating a computer program with structured constructs into its equivalent without structured constructs. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

Guidebook References:

7.1.6.11 Body Stub Generation

Description:

The creation of null bodies for specifications requiring bodies whose corresponding bodies have yet to be defined. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2]

Tools:

Guidebook References:

7.1.6.12 Preamble Generation

. .

Description:

Generating the preamble for a main program that contains parameters. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

Guidebook References:

7.1.6.13 Linking/Loading

Description:

The creation of a load/executable module on the host machine from one or more independently transiated object modules or load modules by resolving cross-references among the object modules, and possibly relocating elements. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

[Linker	5.3.5,
Loader	5.3.6]

Guidebook References:

[Power	6.4.21,
@GB: Linking/Loading Checklist	5.1.4]

7.1.6.14 Interpretation

Description:

The translation of a source program into some intermediate data structure, then executing the algorithm by carrying out each operation given in the intermediate structure. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:	
[Interpreter	5.3.7]

Guidebook References:

7.1.7 Synthesis

Description:

The generation of programs according to predefined rules from a program specification or intermediate language. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.1.7.1 Design Generation

Description:

The translation or interpretation used to construct program designs.

Cross References:

Life Cycle Phases:	
[Preliminary Design	4.4.2,
Detailed Design	4.5.2]

Tools:

Guidebook References:

7.1.7.2 Requirements Reconstruction

Description:

The extraction of software requirements statements from a program design language.

Cross References:

Life Cycle Phases:

[Preliminary Design	4.4.2,
Detailed Design	4.5.2,
Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

Guidebook References:

7.1.7.3 Program Generation

Description:

The translation or interpretation used to construct computer programs (e.g., language translator generator, syntax analyzer generator, code generator generator, environment definition generator, user interface generator, etc.). [@Kean 1985]

Cross References:

Life Cycle Phases:
[Coding And Unit Testing 4.6.2]

Tools:

Guidebook References:

7.1.7.4 Source Reconstruction

Description:

The extraction of lexical and structure attributes from an intermediate language. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

Guidebook References:

7.1.7.5 Decompilation

Description:

The translation of machine code sequences into a higher level procedure-oriented language (i.e., back into the same language from which the machine code sequences were generated). [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

Guidebook References:

7.1.7.6 Disassembling

Description:

The translation of machine code sequences into assembly language sequences. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.2,
CSC Integration And Testing	4.7.2,
CSCI Testing	4.8.2]

Tools:

Guidebook References:

7.2 MANAGEMENT

Description:

Features that aid the management or control of system/software development. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.2.1 Information Management

Description:

The organization, accessing, modification, dissemination, and processing of information that is associated with the development of a software system. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.2.1.1 Data Base (Object) Management

Description:

Managing a collection of interrelated data stored together with controlled redundancy to serve one or more applications and independent of the programs using the data. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Data Base Manager 5.1.9]

Guidebook References:

[Power 6.4.21,
@GB: Data Base Management Checklist 5.1.5]

7.2.1.2 Documentation Management

Description:

The development and control of software documentation. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Document Manager 5.4.1]

Guidebook References:

7.2.1.3 File Management

Description:

Providing and controlling access to files associated with the development of software. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[File Manager 5.1.6]

Guidebook References:

[Power 6.4.21,
@GB: File Management Checklist 5.1.6]

7.2.1.4 Electronic Mail

Description:

The process of receiving/sending messages from/to other system users. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Electronic Mail 5.5.4]

Guidebook References:

[Power 6.4.21,
@GB: Electronic Mail Checklist 5.1.7]

7.2.1.5 Electronic Conferencing

Description:

The concurrent on-line correspondence between two or more users. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Electronic Conferencing 5.5.6]

Guidebook References:

7.2.1.6 Specification Management

Description:

The control of requirements specifications. Specification management features are somewhat methodology dependent because associated with them are requirements languages with formal procedures for their use. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Software Requirements Analysis	4.3.1,
Preliminary Design	4.4.1,
Detailed Design	4.5.1]

Tools:

Guidebook References:

7.2.1.7 Program Library Management

Description:

The creation, manipulation, display, and deletion of the various components of a program library. A program library is a repository for all program information (e.g., source programs, object programs, executable programs, documentation, data, etc.). [@Kean 1985]

Cross References:

Life Cycle Phases:

[Software Requirements Analysis	4.3.1,
Preliminary Design	4.4.1,
Detailed Design	4.5.1,
Coding And Unit Testing	4.6.1,
CSC Integration And Testing	4.7.1,
CSCI Testing	4.8.1]

Tools:

[Program Library Manager	5.3.1]

Guidebook References:

[Power 6.4.21,
@GB: Program Library Management Checklist 5.1.8]

7.2.1.8 Test Data Management

Description:

The development and control of software test data. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.1,
CSC Integration And Testing	4.7.1,
CSCI Testing	4.8.1]

Tools:

Guidebook References:

7.2.1.9 Evaluation Results Management

Description:

The cataloguing and maintenance of the results from the operational test and evaluation of the product. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Operational Testing And Evaluation 4.10.1]

Tools:

Guidebook References:

7.2.1.10 Performance Monitoring

Description:

The monitoring of the performance characteristics of the finished product. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

[Power 6.4.21,
@GB: Performance Monitoring Checklist 5.1.9]

7.2.2 Project Management

Description:

The management of a system/software development project. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.2.2.1 Cost Estimation

Description:

The process of determining the amount of labor necessary for the completion of a task, the amount and potential costs of computer time required, etc., prior to and during a project's lifetime. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Global	4.12.1]

Tools:

[Cost Estimator	5.2.1]

Guidebook References:

7.2.2.2 Quality Specification

Description:

The selection and prioritization of quality factors required for a given application; availability of techniques to apply trade-off analyses (quality factor vs. quality factor, and quality factor vs. cost) and quantify quality requirements. Potential quality factors include: efficiency, integrity, reliability, survivability, usability, correctness, maintainability, verifiability/testability, expandability/flexibility, interoperability, reusability, and transportability. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Quality Analyzer 5.2.2]

Guidebook References:

7.2.2.3 Scheduling

Description:

The process of identifying tasks, products to be delivered, delivery dates, personnel needed to complete tasks, etc. for a development project. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Scheduling 5.2.3,
Calendar 5.5.7]

Guidebook References:

[Power 6.4.21,
@GB: Scheduling Checklist 5.1.10]

7.2.2.4 Work Breakdown Structure

Description:

The enumeration of all work activities in hierarchic refinements of detail that divides work to be done into short, manageable tasks with quantifiable inputs, outputs, schedules, and assigned responsibilities. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Work Breakdown Structure 5.2.4]

Guidebook References:

7.2.2.5 Resource Estimation

Description:

The estimation of resources attributed to an entity. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Software Requirements Analysis	4.3.1,
Preliminary Design	4.4.1,
Detailed Design	4.5.1,
Coding And Unit Testing	4.6.1,
CSC Integration And Testing	4.7.1,
CSCI Testing	4.8.1]

Tools:

[Resource Estimator	5.2.5]

Guidebook References:

7.2.2.6 Tracking

Description:

The tracking of the development of an entity through the software life cycle. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Software Requirements Analysis	4.3.3,
Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.4,
CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3]

Tools:

[Tracking	5.2.6]

Guidebook References:

[Power 6.4.21,
@GB: Tracking Checklist 5.1.11]

7.2.2.7 Configuration Management

Description:

The establishment of baselines for configuration items, the control of changes to these baselines, and the control of releases to the operational environment. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Configuration Manager 5.2.7]

Guidebook References:

[Power 6.4.21,
@GB: Configuration Management Checklist 5.1.12]

7.2.2.8 Quality Assessment

Description:

The use of field data to determine the achieved level of quality in deployed software systems. A verification that the specified quality requirements have been met. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:
[Quality Analyzer 5.2.2]

Guidebook References:

7.2.3 Computer System Management

Description:

The management of hardware/software architectures to support the life cycle software engineering environment. Such services include: creating, scheduling, and removing tasks; switching the processor among tasks; sending messages between tasks; providing direct and import/export access; managing distributed systems; sending files from one host machine to another on a network, etc. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.2.3.1 Command Language Processing

Description:

The processing of command language constructs into functions performed by the operating system. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Command Language Processor 5.1.1]

Guidebook References:

7.2.3.2 Input/Output Support

Description:

The services for accessing standard I/O devices such as disks, tapes, terminals, and printers from within a program.

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Runtime Library 5.3.8]

Guidebook References:

[Power 6.4.21,
@GB: Input/Output Support Checklist 5.1.13]

7.2.3.3 Kernel

Description:

The functions which provide access for tools and application programs to services of the native operating system.

Cross References:

Life Cycle Phases:
[Global 4.12.1]

Tools:
[Security System 5.1.3]

Guidebook References:

[Completeness 6.4.9, @GB: CIVC 8.2]

7.2.3.4 Math/Statistics

Description:

The services for supporting standard math and statistical operations.

Cross References:

Life Cycle Phases:
[Global 4.12.3]

Tools:
[Runtime Library 5.3.8,
Spreadsheet 5.5.1,
Calculator 5.5.2]

Guidebook References:

7.2.3.5 Runtime Environment

Description:

The functions that can be expected to be found in the runtime libraries for Ada implementations. It should be noted that the dividing line between the predefined runtime support library on the one hand and the conventions and data structures of a compiler on the other hand is not always obvious. One Ada implementation may use a predefined routine to implement a particular language feature, while another implementation may realize the same feature through conventions for the executable code. Not addressed here are elements of a runtime library that exist because of special characteristics of the underlying computing resource. [@ARTEWG 1986]

Cross References:

Life Cycle Phases:
[Global 4.12.2]

Tools:
[Job Scheduler 5.1.4,
Resource Controller 5.1.5,
Runtime Library 5.3.8]

Guidebook References:

[Power 6.4.21,
@GB: Runtime Environment Checklist 5.1.14]

7.2.3.6 Import/Export

Description:

The services for communicating objects between various computer systems or networks.

Cross References:

Life Cycle Phases:

[System Integration and Testing	4.9.2, 4.9.3,
Operational Testing and Evaluation	4.10.3]

Tools:

[Archive, Backup, and Retrieval System	5.1.2,
Import/Export System	5.1.7]

Guidebook References:

[Power	6.4.21,
@GB: Import/Export Checklist	5.1.15]

7.3 ANALYSIS

Description:

The features that provide an examination of a substantial whole to determine both qualitative and quantitative properties. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.3.1 Static Analysis

Description:

Static analysis features specify operations on the subject without regard to the executability of the subject. They describe the manner in which the subject is analyzed. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.3.1.1 Comparison

Description:

The determination and assessment of similarities between two or more items. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Global	4.12.3]
Tools:	
[Comparator	5.6.1]

Guidebook References:

7.3.1.2 Spelling Checking

Description:

The identification of incorrectly spelled words. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Global 4.12.3]

Tools:
[Spell Checker 5.4.3]

Guidebook References:

7.3.1.3 Data Flow Analysis

Description:

The analysis of the formal requirements statements to determine interface consistency and data availability. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Software Requirements Analysis	4.3.3,
Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:

[Data Flow Analyzer	5.6.2]

Guidebook References:

7.3.1.4 Functional Analysis

Description:

The analysis of formally stated requirements to determine their consistency and completeness. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Software Requirements Analysis 4.3.3]

Tools:
[Functional Analyzer 5.6.4]

Guidebook References:

7.3.1.5 Interface Analysis

Description:

The checking of interfaces between program elements for consistency and adherence to predefined rules and/or axioms. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Software Requirements Analysis	4.3.3,
Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:

[Interface Analyzer	5.6.4]

Guidebook References:

7.3.1.6 Traceability Analysis

Description:

The checking for internal consistency within the software requirements specifications. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Software Requirements Analysis 4.3.3]

Tools:
[Traceability Analyzer 5.6.5]

Guidebook References:

7.3.1.7 Testability Analysis

Description:

The quantitative measurement of the extent to which a component facilitates the establishment of verification criteria and supports evaluation of its performance.

Cross References:

Life Cycle Phases:
[Software Requirements Analysis 4.3.3]

Tools:
[Testability Analyzer 5.6.6]

Guidebook References:

7.3.1.8 Test Condition Analysis

Description:

The analysis of formal requirements language statements to determine data values to be examined and mechanisms to be used in the verification of test results. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Software Requirements Analysis 4.3.3]

Tools:
[Test Condition Analyzer 5.6.7]

Guidebook References:

7.3.1.9 Quality Analysis

Description:

The quantitative measurement of specified quality factors for use during the evaluation of software products (and prediction of software quality) at key milestones during development. Factors to be analyzed include: efficiency, integrity, reliability, survivability, usability, correctness, maintainability, verifiability/testability, expandability/flexibility, interoperability, reusability, and transportability. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Software Requirements Analysis	4.3.3,
Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3]

Tools:	
[Quality Analyzer	5.6.8]

Guidebook References:

7.3.1.10 Complexity Measurement

Description:

The determination of how complicated an entity (e.g., routine, program, system, etc.) is by evaluating some number of associated characteristics. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:	
[Complexity Measurer	5.6.9]

Guidebook References:

7.3.1.11 Correctness Checking

Description:

The determination of agreement between the component as coded and the programming specification (algorithmic correctness).

Cross References:

Life Cycle Phases:

Tools:
[Correctness Checker 5.6.10]

Guidebook References:

7.3.1.12 Completeness Checking

Description:

The assessment of whether or not an entity has all its parts present and if those parts are fully developed. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:

[Completeness Checker	5.6.11]

Guidebook References:

7.3.1.13 Consistency Checking

Description:

The determination of whether or not an entity is internally consistent in the sense that it is consistent with its specification. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:	
[Consistency Checker	5.6.12]

Guidebook References:

7.3.1.14 Reusability Analysis

Description:

The quantitative measurement of specified reusability factors for use during the evaluation of software products (and prediction of software reusability).

Cross References:

Life Cycle Phases:

Tools:
[Reusability Analyzer 5.6.13]

Guidebook References:

7.3.1.15 Syntax And Semantics Checking

Description:

The detection of errors in the syntax and semantics of a formal language.

Cross References:

Life Cycle Phases:

Tools:

[Syntax-Directed Editor	5.3.2,
Word Processor	5.4.2,
Syntax And Semantics Checking	5.6.14]

Guidebook References:

7.3.1.16 Reachability Analysis

Description:

The detection of sections of code that cannot be executed because of the structure of the code unit that contains it.

Cross References:

Life Cycle Phases:

Tools:
[Reachability Analyzer 5.6.15]

Guidebook References:

7.3.1.17 Cross Reference

Description:

The referencing of entities to other entities by logical means. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:

[Cross Referencer	5.6.16]

Guidebook References:

7.3.1.18 Maintainability Analysis

Description:

The quantitative measurement of specified maintainability factors for use during the evaluation of software products (and prediction of software maintainability).

Cross References:

Life Cycle Phases:

Tools:
[Maintainability Analyzer 5.6.17]

Guidebook References:

7.3.1.19 Invocation Analysis

Description:

The analysis of a system/software design for the purpose of determining calling relationships between elements. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3]
Tools:	
[Invocation Analyzer	5.6.18]

Guidebook References:

7.3.1.20 Scanning

Description:

The examination of entities sequentially to identify key areas or structure. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:

[Scanner	5.6.19]

Guidebook References:

7.3.1.21 Structured Walkthrough

Description:

The interactive display of source code (or its design) with the capability for branching to subordinate program modules/units. This feature automates code reading such that a walkthrough of a computer program can be performed on a video terminal. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3]
Tools:	
[Structured Walkthrough Tool	5.6.20]

Guidebook References:

7.3.1.22 Auditing

Description:

The conducting of an examination to determine whether or not predefined rules have been followed. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:

[Auditor	5.6.21]

Guidebook References:

7.3.1.23 Error Checking

Description:

The determination of discrepancies, their importance, and/or their cause. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:

[Error Checker	5.6.22]

Guidebook References:

7.3.1.24 Statistical Analysis

Description:

The performance of statistical data collection and analysis. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Coding And Unit Testing 4.6.3]

Tools:
[Statistical Analyzer 5.6.23]

Guidebook References:

7.3.1.25 Statistical Profiling

Description:

The analysis of a computer program to determine statement types, number of occurrences of each statement type, and the percentage of each statement type in relation to the complete program. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Coding And Unit Testing 4.6.3,
CSC Integration And Testing 4.7.3]

Tools:
[Statistical Profiler 5.6.24]

Guidebook References:

7.3.1.26 Structure Checking

Description:

The detection of structural flaws within a program (e.g., improper loop nestings, unreferenced labels, unreachable statements, and statements with no successors). Flaws detected by this function are not illegal or erroneous, but do constitute bad style. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]
Tools:	
[Structure Checker	5.6.25]

Guidebook References:

7.3.1.27 Type Analysis

Description:

The evaluation of whether or not the domain of values attributed to an entity are properly and consistently defined. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:

[Type Analyzer	5.6.26]

Guidebook References:

7.3.1.28 Units Analysis

Description:

The determination of whether or not the units or physical dimensions attributed to an entity are properly defined and consistently used. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:	
[Units Analyzer	5.6.27]

Guidebook References:

7.3.1.29 I/O Specification Analysis

Description:

The analysis of the input and output specifications in a program usually for the generation of test data. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:	
[*I/O Specification Analyzer*	5.6.28]

Guidebook References:

7.3.1.30 Sizing Analysis

Description:

The quantitative measurement of the maximum amount of primary (and secondary) storage required to run a program.

Cross References:

Life Cycle Phases:

Tools:
[Sizing Analyzer 5.6.29]

Guidebook References:

7.3.2 Dynamic Analysis

Description:

Dynamic analysis features specify operations that are determined during or after execution takes place. Dynamic analysis features differ from those classified as static by virtue of the fact that they require some form of symbolic or machine execution. They describe the techniques used by the tool to derive meaningful information about a program's execution behavior. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:

Guidebook References:

7.3.2.1 Requirements Simulation

Description:

The execution of code enhanced requirements statements to examine functional interfaces and performance. [@Kean 1985]

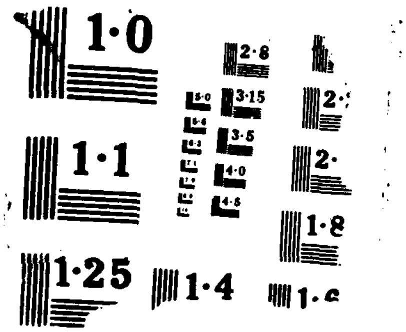
Cross References:

Life Cycle Phases:	
[System Requirements Analysis	4.2.3,
Software Requirements Analysis	4.3.3]
Tools:	
[Requirements Simulator	5.7.1]

Guidebook References:

AD-A193 389 E&V (EVALUATION AND VALIDATION) MANUAL VERSION 10(U) 4/4
ADA JOINT PROGRAM OFFICE ARLINGTON VA 29 DEC 87
TASC-TR-5234-3

UNCLASSIFIED F/G 12/5 NL


1·0
2·8
3·15
3·5
4·0
4·5
1·1
1·8
1·25
1·4

7.3.2.2 Requirements Prototyping

Description:

The rapid construction of critical functions of a system early in the life cycle for the purpose of understanding the requirements. During this activity, these results are intended to be thrown away. [@Kean 1985]

Cross References:

Life Cycle Phases:

[System Requirements Analysis	4.2.3,
Software Requirements Analysis	4.3.3]

Tools:

[Requirements Prototype	5.7.2]

Guidebook References:

[Power 6.4.21,
@GB: Requirements Prototyping Checklist 5.1.16]

7.3.2.3 Simulation And Modeling

Description:

The representation of selected characteristics of the behavior of one physical or abstract system by another system (e.g., the representation of physical phenomena by means of operations performed by a computer system, the representation of operations of a computer system by those of another computer system, etc.). [@Kean 1985]

Cross References:

Life Cycle Phases:
[Preliminary Design 4.4.3,
Detailed Design 4.5.3]

Tools:
[Simulation And Modeling Tools 5.7.3]

Guidebook References:

[Power 6.4.21,
@GB: Simulation and Modeling Checklist 5.1.17]

7.3.2.4 Design Prototyping

Description:

The rapid construction of critical functions of a system early in the life cycle for the purpose of understanding the requirements. During this activity, the results will be retained for the purpose of incrementally developing the product. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Preliminary Design 4.4.3,
Detailed Design 4.5.3]

Tools:
[Design Prototype 5.7.4]

Guidebook References:

7.3.2.5 Debugging

Description:

The process of locating, analyzing, and correcting suspected faults in a program. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3]

Tools:

[Debugger	5.7.5]

Guidebook References:

[Power	6.4.21,
@GB: Debugging Checklist	5.1.18]

7.3.2.6 Executable Assertion Checking

Description:

The checking of user-embedded statements that assert relationships between elements of a program. An assertion is a logical expression that specifies a condition or relation among the program variables. Checking may be performed with symbolic or run-time data. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]

Tools:

[Executable Assertion Checker	5.7.6]

Guidebook References:

[Power	6.4.21,	
@GB: Executable Assertion Checking Checklist		5.1.19]

7.3.2.7 Constraint Evaluation (Contention)

Description:

The generation and/or solution of path input or output constraints for determining test input or proving programs correct. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]
Tools:	
[Constraint Evaluator	5.7.7]

Guidebook References:

7.3.2.8 Coverage/Frequency Analysis

Description:

The determination and assessment of measures associated with the invocation of program structural elements to determine the adequacy of a test run. Coverage analysis is useful when attempting to execute each statement, branch, path, or program. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3]

Tools:

[Coverage/Frequency Analyzer	5.7.8]

Guidebook References:

7.3.2.9 Mutation Analysis

Description:

The application of test data to a program and its "mutants" (i.e., programs that contain one or more likely errors) in order to determine test data adequacy. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Coding And Unit Testing 4.6.3,
CSC Integration And Testing 4.7.3]

Tools:
[Mutation Analyzer 5.7.9]

Guidebook References:

[Power 6.4.21,
@GB: Mutation Analysis Checklist 5.1.20]

7.3.2.10 Testing

Description:

The identification of necessary test cases to accomplish testing objectives, the generation of actual test data for each test case to support both structural and functional testing, the execution of the test cases, and the analysis of the test results.

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3]

Tools:

[Testing Analyzer	5.7.10]

Guidebook References:

[Power 6.4.21,
@GB: Testing Checklist 5.1.21]

7.3.2.11 Regression Testing

Description:

The rerunning of test cases which a program has previously executed correctly in order to detect errors spawned by changes or corrections made during software development and maintenance. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3]

Tools:

[Regression Testing Analyzer	5.7.11]

Guidebook References:

7.3.2.12 Resource Utilization

Description:

The analysis of resource utilization associated with system hardware or software. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3]

Tools:

[Resource Utilization Analyzer	5.7.12]

Guidebook References:

7.3.2.13 Emulation

Description:

The imitation of all or part of one computer system by another, primarily by hardware, so that the imitating computer system accepts the same data, executes the same programs, and achieves the same results as the imitated system. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3]

Tools:

[Emulator	5.7.13]

Guidebook References:

[Power 6.4.21,
@GB: Emulation Checklist 5.1.22]

7.3.2.14 Timing

Description:

The reporting of actual CPU, wall-clock, or other times associated with parts of a program. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3]

Tools:

[Timing Analyzer	5.7.14]

Guidebook References:

[Power	6.4.21,
@GB: Timing Checklist	5.1.23]

7.3.2.15 Tuning

Description:

The activity of optimizing parts of a program which account for significant amounts of execution time. The optimization follows the determination of a "performance profile" which is the identification of the parts of a program which use the most execution time.

Cross References:

Life Cycle Phases:
[Coding And Unit Testing 4.6.3,
CSC Integration And Testing 4.7.3,
CSCI Testing 4.8.3]

Tools:
[Tuning Analyzer 5.7.15]

Guidebook References:

[Power 6.4.21,
@GB: Tuning Checklist 5.1.24]

7.3.2.16 Reliability Analysis

Description:

The determination of the ability of an item to perform a required function under stated conditions for a stated period of time. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:
[Reliability Analyzer 5.7.16]

Guidebook References:

7.3.2.17 Real Time Analysis

Description:

The determination of the behavior of a program which must recognize, interpret, and respond to external, asynchronous events at speeds which allow the system to handle all such events in a "reasonable" amount of time.

Cross References:

Life Cycle Phases:

Tools:
[Real Time Analyzer 5.7.17]

Guidebook References:

[Power 6.4.21,
@GB: Real Time Analysis Checklist 5.1.25]

7.3.3 Formal Verification

Description:

The use of rigorous mathematical techniques to prove the consistency between an algorithmic solution and a rigorous, complete specification of the intent of the solution. [@Kean 1985]

Cross References:

Life Cycle Phases:

[Preliminary Design	4.4.3,
Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3]

Tools:

[Formal Verification System	5.7.18]

Guidebook References:

7.3.4 Symbolic Execution

Description:

The reconstruction of the logic and computations along a program path by executing the path with symbolic rather than actual values of data. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[Detailed Design	4.5.3,
Coding And Unit Testing	4.6.3,
CSC Integration And Testing	4.7.3]
Tools:	
[Symbolic Execution System	5.7.19]

Guidebook References:

7.3.5 Problem Report Analysis

Description:

The analysis of problem reports for the purpose of determining the validity of the reported problem and a corrective action. [@Kean 1985]

Cross References:

Life Cycle Phases:	
[CSC Integration And Testing	4.7.3,
CSCI Testing	4.8.3,
System Integration And Testing	4.9.3,
Operational Testing And Evaluation	4.10.3]

Tools:	
[Problem Report Analyzer	5.2.8]

Guidebook References:

7.3.6 Change Request Analysis

Description:

The analysis of change requests to determine the necessity of the change, technical/economic impacts, and approach to accomplishing the change. [@Kean 1985]

Cross References:

Life Cycle Phases:

Tools:
[Change Request Analyzer 5.2.9]

Guidebook References:

7.3.6.1 Change Impact Analysis

Description:

The ability to determine, for a proposed support/enhancement operation, the impact of proposed changes to the software system. For example, changes could be specified at the requirements, design, or code levels and, utilizing the traceability mechanisms which link elements of various life cycle activities, the impact(s) of the changes could be identified. Impact(s) would include those to requirements, design, code, test cases/test data, and associated documentation. [@Kean 1985]

Cross References:

Life Cycle Phases:
[Change Requirements 4.11.3]

Tools:

Guidebook References:

THIS PAGE LEFT INTENTIONALLY BLANK

APPENDIX A
CITATIONS

[ACVC 1986] "Ada Validation Policies and Procedures," Ada Joint Program Office, Draft, 24 January 1986.

[ARTEWG 1986] "A Canonical Model and Taxonomy of Ada Runtime Environments," Proposed by the Ada Runtime Environments Working Group of SIGAda, 13 November 1986.

[Barstow and Shrobe 1981] D.R. Barstow and H.E. Shrobe, "Observations on Interactive Programming Environments," [@IEEE 1981], 286-301, 1981.

[Buxton 1980] [@DoD 1980].

[CAIS] [@DoD 1986].

[Castor 1983] V.L. Castor, "Criteria for the Evaluation of ROLM Corporation's Ada Work Center," Wright-Patterson AFB, 10 January 1983.

[DACS 1979] The DACS Glossary, A Bibliography of Software Engineering Terms, RADC/COED, Griffiss AFB, NY, October 1979.

[DoD-STD-2167] Defense System Software Development, U.S. Department of Defense, 4 June 1985.

[DoD 1980] J.N. Buxton, "Requirements for Ada Programming Support Environments -- STONEMAN," U.S. Department of Defense, February 1980.

[DoD 1983] ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language, U.S. Department of Defense, 17 February 1983.

[DoD 1986] "Military Standard Common Ada Programming Support Environment (APSE) Interface Set (CAIS)," U.S. Department of Defense, DoD-STD-1838, 9 October 1986.

[E&V Classification] "E&V Classification Schema Report", AFWAL/AAAF, Wright-Patterson AFB, OH, Version 1.0, 15 June 1987.

[E&V Guidebook] [@GB].

[E&V Plan] [@E&V Report 1984: E&V Plan A.].

[E&V Report 1984] Evaluation and Validation (E&V) Team Public Report, Volume I, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, 30 November 1984.

[E&V Report 1987] Evaluation and Validation (E&V) Team Public Report, Volume III, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, in preparation.

[E&V Requirements][@E&V Report 1984: E&V Requirements B.].

[E&V Workshop 1984] V.L. Castor, et al., "Ada Programming Support Environment (APSE) Evaluation and Validation (E&V) Workshop Report," Institute for Defense Analysis, December 1984.

[GB] "E&V Guidebook," AFWAL/AAAF, Wright-Patterson AFB, OH, Version 1.0, in preparation.

[Gutz et al 1981] S. Gutz, A.I. Wasserman, and M.J. Spier, "Personal Development Systems for the Professional Programmer," Computer, April 1981.

[Henderson 1987] P.B. Henderson, "Software Development/Programming Environments," ACM Software Engineering Notes, Volume 12, Number 1, January 1987.

[Houghton 1983] R.C. Houghton, Jr., "A Taxonomy of Tool Features for the Ada Programming Support Environment (APSE)," U.S. Department of Commerce, National Bureau of Standards, December 1982, Issued February 1983.

[Houghton and Wallace 1987] R.C. Houghton, Jr. and D.R. Wallace, "Characteristics and Functions of Software Engineering Environments," ACM Software Engineering Notes, Volume 12, Number 1, January 1987.

[IEEE 1981] Tutorial: Software Development Environments, ed. A.I. Wasserman, IEEE Catalog No. EHO 187-5, 1981.

[IEEE 1983] "IEEE Standard Glossary of Software Engineering Terminology," IEEE Std 729-1983.

[IEEE Expert 1986] IEEE Expert, Winter 1986.

[Kean 1985] E.S. Kean and F.S. Lamonica, "A Taxonomy Of Tool Features For A Life Cycle Software Engineering Environment," Rome Air Development Center, Griffiss AFB, June 1985.

[Lehman and Turski 1987] M.M. Lehman and W.M. Turski, "Essential Properties of IPSEs," ACM Software Engineering Notes, Volume 12, Number 1, January 1987.

[Lyons and Nissen 1986] "Selecting an Ada Environment," eds. T.G.L. Lyons and J.C.D. Nissen, Ada-Europe Environments Working Group, Cambridge University Press, 1986.)

[NBS Taxonomy] [@Houghton 1983].

[McDermid and Ripken 1984] J. McDermid and K. Ripken, "Life Cycle Support in the Ada Environment," Cambridge University Press, 1984.

[McDermid 1985] Integrated Project Support Environments, ed. J. McDermid, Peter Peregrinus (IEEE), 1985.

[McKay 1987] McKay, C.W., "A Proposed Framework for the Tools and Rules to Support the Life Cycle of the Space Station Program," Proceedings of the IEEE Compass '87 Conference, June 1987.

[Notkin and Habermann 1981] D.S. Notkin and A.N. Habermann, "Software Development Environment Issues as Related to Ada," [@IEEE 1981], 107-133, 1981.

[PIWG 1987] Ada Slices, Official Newsletter of the ACM SIGAda User's Committee Performance Issues Working Group, "PIWG," 5 March 1987.

[RADC 1985] T.P. Bowen, G.B. Wigle, and J.T.Tsai, "Specification of Software Quality Attributes Software Quality Specification Guidebook," Rome Air Development Center, Griffiss AFB, RADC-TR-85-37, Volume II (of three), February 1985.

[SEE Taxonomy] [@Kean 1985].

[STONEMAN] [@DoD 1980].

[Texas Instruments 1985] The APSE Interactive Monitor, Texas Instruments, Slide Presentation to the E&V Team, 5 September 1985.

[Wasserman 1981a] A.I. Wasserman, "The Ecology of Software Development Environments," [@IEEE 1981].

[Wasserman 1981b] A.I. Wasserman, "Toward Integrated Software Development Environments," [@IEEE 1981].

[Zualkernan et al 1986] I. Zualkernan, W.T. Tsai, and D. Volovik, "Expert Systems and Software Engineering: Ready for Marriage?," IEEE Expert, Winter, 1986.

THIS PAGE LEFT INTENTIONALLY BLANK

APPENDIX B
GLOSSARY

B.1 ACRONYMS AND ABBREVIATIONS

This section provides a list of all acronyms and abbreviations used in this manual, with the associated meanings.

APSE	Ada Programming Support Environment
ARTEWG	Ada Runtime Environments Working Group (SIGAda)
CAIS	Common APSE Interface Set
CBD	Commerce Business Daily
CRISD	Computer Resources Integrated Support Document
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSDM	Computer System Diagnostic Manual
CSOM	Computer System Operator's Manual
DACS	Data and Analysis Center for Software
DBDD	Data Base Design Document
DoD	Department of Defense
ECP	Engineering Change Proposal
E&V	Evaluation and Validation
FSM	Firmware Support Manual
GB	E&V Guidebook
HWCI	Hardware Configuration Item
IDD	Interface Design Document
IEEE	Institute of Electrical and Electronics Engineers
IPSE	Integrated Project Support Environment
IRS	Interface Requirements Specification
LLCSC	Lower-level Computer Software Component
MCCS	Mission-Critical Computer System
OCD	Operational Concept Document
PIWG	Performance Issues Working Group (SIGAda)
PSE	Programming Support Environment
RADC	Rome Air Development Center
RFP	Request for Proposal
RM	E&V Reference Manual
RMGB	Reference Manual/Guidebook
SCMP	Software Configuration Management Plan
SCN	Specification Change Notice

SDDD	Software Detailed Design Document
SDE	Software Development Environment
SDF	Software Development File
SDP	Software Development Plan
SEE	Software Engineering Environment
SIGAda	Special Interest Group for Ada of the Association for Computing Machinery (ACM)
SOW	Statement of Work
SPM	Software Programmer's Manual
SPS	Software Product Specification
SQEP	Software Quality Evaluation Plan
SRS	Software Requirements Specification
SSPM	Software Standards & Procedures Manual
SSS	System/Segment Specification
STD	Software Test Description
STLDD	Software Top Level Design Document
STP	Software Test Plan
STPR	Software Test Procedure
STR	Software Test Report
SUM	Software User's Manual
TLCSC	Top-level Computer Software Component
VDD	Version Description Document
WBS	Work Breakdown Structure

APPENDIX C
FORMAL GRAMMAR

This appendix specifies sections of the Reference Manual and Guidebook (RMGB) as a formal grammar. The sections include chapters four through seven of the Reference Manual (RM), all explicit references, the table of contents, the composite index, and the reference appendix. The specification is presented as a partitioned grammar for convenience.

(The grammar is presented in a modified Backus-Naur form. Brackets represent optionality when alone, and may be marked by an asterisk '*' to denote 0–N instances of the production, or by a sharp '#' to denote 1–N instances. Angle brackets denote comments in place of productions which are too elaborate to express here. All terminals of the grammar are expressed as quoted literals, or composite literals based on characters and character strings.)

C.1 FORMAL REFERENCES

Throughout the RMGB, whenever formal references are made, a single consistent set of grammar rules are used. This includes reference from one volume to the other, reference from one section in a volume to another section in the same document, and reference to documents outside the RMGB.

```
reference_list  ::= "[" references  [";"  references  ]* "]"

references      ::= reference [ ","  reference ]*

reference       ::= "@" phrase [":" [ phrase ] [ designator_list ]

                    | [ phrase ]  designator_list

phrase          ::= <text lacking special characters>
```

```
designator_list  ::=  designator [ "," designator ]*

designator       ::=  leads "." |  leads [ "." digits ]*

leads            ::=  digits | caps

digits           ::=  ('0'-'9')

caps             ::=  ('A'-'Z')
```

C.2 INDEXES

The index format of the RMGB builds on the reference specifications above, and is derived from MIL-STD-1815A.

```
index            ::=  [ entry ]#

entry            ::=  topic [ redirector ] [ reference_list ]

topic            ::=  phrase [ "(" designator_list ")" ]

redirector       ::=  "[" "see" [ "also" ] ":' phrase_list "]"

phrase_list      ::=  phrase [ "," phrase ]*
```

C.3 FORMAL CHAPTERS

Those chapters of the RM which are derived from the classification schema are formally defined here.

C.3.1 Chapter Components

The following rules define the components which are used to compose formal chapter entries.

```
descriptor       ::=  designator phrase [description]

description      ::=  "Description:" text
```

```
text            ::=  < prose text >

functions       ::=  "Functions:" reference_list

deliverables    ::=  "Deliverables:" reference-list

gb_references   ::=  "Guidebook:" reference_list

life-cycle-phases ::= "Life Cycle Phases:" reference_list

tools           ::=  "Tools:" reference_list

quality_factors ::=  "Software Quality Factors:"

                     reference_list

acquisition_concern ::= "Acquisition Concern:"

                         reference_list

software_criteria ::=  "Software-Oriented Criteria:"
                         reference_list

characteristics ::=  "Application/Environment Characteristics:"
                     reference_list

complementary_factors ::=  "Complementary Software Quality Factors:"
                           reference_list

cooperating_criteria  ::= "Cooperating Criteria:"
                          reference_list

conflicting_criteria   ::=  "Conflicting Criteria:"
                            reference_list

beneficial_factors   ::=  "Beneficial Quality Factors:"
                          reference_list

adverse_factors      ::=  "Adverse Quality Factors:"
                          reference_list
```

C.3.2 Chapter Entries

Each numbered section of the formal chapters follows a specific grammar rule. The following rules define the format of each class of chapter entries.

```
life_cycle_phase    ::= descriptor management transformation analysis
                        oper_and_support

management       ::= descriptor functions deliverables

transformation   ::= descriptor functions deliverables

analysis         ::= descriptor functions deliverables

oper_and_support ::= descriptor functions deliverables

apse             ::= descriptor [ toolset ]*

toolset          ::= descriptor [ tool ]*

tool             ::= descriptor functions

attribute        ::= descriptor [ concern ]* [factor ]*
                                 [ criterion ]*

concern          ::= descriptor quality factors

factor           ::= descriptor acquisition_concern
                     software_criteria characteristics
                     complementary_factors cooperating_criteria
                     conflicting_criteria
criterion        ::= descriptor quality_factors
                     beneficial_factors adverse_factors
                     gb_references

function         ::= descriptor life_cycle_phases tools
                     gb_references
```

C.3.3 Formal Chapter Ordering

The formal portion of the RM is found in chapters four through seven. Each of the classes of chapter entries is found in a distinct chapter.

```
formal_chapters   ::= [ life_cycle phase ]*
                      [ apse ]
                      [ attribute ]*
                      [ function ]*
```

C.4 TABLE OF CONTENTS

The table of contents shares some features with the rest of the formal aspects of the RMGB.

```
table_of_contents  ::= [ chapter ]* index
chapter            ::= designator phrase designator_page
designator_page    ::= designator "-" digits
index              ::= "Index" digits
```

C.5 CITATIONS

The citations are found in Appendix A, and have a formal structure as defined in the following grammar. The (semantic) form of citation text is taken from the standard for IEEE Software Magazine.

```
citations   ::= [ citation ]*
citations   ::= key body "."
key         ::= "[" text "]"
body        ::= text | key
```

THIS PAGE LEFT INTENTIONALLY BLANK

INDEX

THIS PAGE LEFT INTENTIONALLY BLANK

END
DATE
FILMED
7-88
DTIC